Object Recognition

Modelling and the Interface to a Control Strategy for Matching

Peter Boros

Object Recognition

Modelling and the Interface to a Control Strategy for Matching

Thesis for the degree of doktor ingeniør

Trondheim, September 2007

Norwegian University of Science and Technology
Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Computer and Information Science

NTNU
Innovation and Creativity

NTNU
Norwegian University of Science and Technology

Thesis for the degree of doktor ingeniør

Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Computer and Information Science

ISBN 978-82-471-4364-3 (printed ver.)
ISBN 978-82-471-4378-0 (electronic ver.)
ISSN 1503-8181

Theses at NTNU, 2007:198

Printed by Tapir Uttrykk

To Thina, Emily and Fredrik

Abstract

A modelling system for object recognition and pose estimation is presented in this work, based on approximating the aspect/appearance graph of arbitrary rigid objects for a spherical viewing surface using simulated image data. The approximation is achieved by adaptively subdividing the viewing sphere starting with an icosahedral tessellation and iteratively decreasing the patch size until the desired resolution is reached. The adaptive subdivision is controlled by both the required resolution and object detail. The decision whether a patch should be divided is based on a similarity measure, which is obtained from applying graph matching to attributed relational graphs generated from image features.

Patches surrounded by similar views are grouped together and reference classes for the aspects are established. The reference classes are indexed by contour types encountered in the views within the group, where the contour types are computed via unsupervised clustering performed on the complete set of contours for all views of a given object.

Classification of an unknown pose is done efficiently via simple or weighted bipartite matching of the contours extracted from the unknown pose to the equivalence classes. The best suggestions are selected by a scoring scheme applied to the match results.

The modelling system is demonstrated by experimental results for a number of objects at varying levels of resolution. Pose estimation results from both synthetic and real images are also presented.

Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgements

I wish to thank my supervisor, professor Richard E Blake for his patience and guidance throughout these years and for not giving up the hope that some day I really will make it to this point.

Thanks to the people at the Department of Computer and Information Science for all the help and support I received since my very first day in Trondheim.

Last, but not least I owe a great deal of thanks to my wife, Thina, and my two children, Emily and Fredrik for being there when I needed them, and for tolerating me so well when I was grouchy after working many late night hours.

Chapter 1

Introduction

Recognising objects is a task humans – and most animals – do all the time. Even small babies are able to recognise such complex objects as the face of their mother. However, when it comes to applying machine tools to this "simple" task, we are facing a number of challenges and far from trivial problems.

First, we need an abstraction – a model – of the objects we are trying to recognise. The model needs to fulfil several criteria – it should be exact, unambiguous and unique; it also needs to be descriptive enough to be able to represent a wide range of objects [129]. In computer vision applications, the choice of the model is often related to the acquisition device, as it is naturally most important to appropriately model the features of the object which we are in fact able to observe. Unfortunately, the features which are easy to model (such as an edge) are not necessarily the ones which are easy to observe (for instance colour of the light reflected from a point on the surface of the object). This requires a mapping from observed features to model features – for example edge detection or contour tracing.

Another problem is raised by the transition between two and three dimensions. In most situations, the object (and the model) is three dimensional, while the observations are two dimensional projections, such as in case of a colour camera. In order to be able to match the observed features to the model features, we need to predict which model features are visible from a given position, and/or estimate the pose of the object based on the observed features.

Pose estimation, based on a reduced number of features or reduced detail is also useful for limiting the search space where more precise matching may be performed using all available features. This process can be assisted by the model, from which the most likely poses of the

object can be determined and tested in the order of decreasing probability. Heuristics such as this play an important role since the matching process can be computationally intensive. Inexact graph matching, for instance is proven to be NP-complete [1].

Partitioning the problem is another way to attack complexity. It is easier to solve two problems of size $\frac{N}{2}$, than one problem of size N, for anything more difficult than $O(N)$ (i.e. linear complexity). If possible, we should rather split a scene into individual objects and recognise the objects each in itself, than trying to treat the whole scene as a single unit. Separating the object from the background is also a necessity of most recognition applications.

In addition to this, the model is often simplified compared to real life, or not 100% accurate. Even if we manage to create a model which is a perfect description of the real objects, there will be some inaccuracy resulting from limitations of the acquisition device. An RGB or a range sensor is only able to measure the specific physical characteristics with a given finite precision. The effect is further increased by environmental conditions such as noise or lighting effects. A recognition system therefore needs to be able to cope with approximate matches and/or take these external conditions into consideration.

For particular object recognition purposes, *aspect graphs* [93] have been introduced as a way of providing a viewer-centric description of objects. Aspect graphs describe the object in terms of stable views and transitions between them (the nodes and edges of the graph). A stable view of the object is a view which is insensitive to small changes in viewing position concerning which surfaces (faces for polygonal objects) are visible. Each stable view of the object determines a viewing cell, which is a subset of the viewing space; and from each point within the viewing cell the same surfaces are visible. There is a transition between two views if the corresponding viewing cells share a common boundary.

Aspect graphs can be extended to include information about the object's appearance under particular conditions in addition to the pure geometric model. These *appearance graphs* [165] can model coloured surfaces, non-uniform illumination, self-shadows, which can change the object appearance without having any change in the aspect.

1.1 Motivation and objectives

Several different algorithms have been developed for computing the aspect graphs of objects (summarised in Section 2.1.4). Many of these however are limited to certain types of objects (polyhedra, solids of revolution, quadric surfaces) and/or are computationally intensive. Several other concerns have been raised about aspect graphs, such as regarding the number of

aspects and their importance, relations to real image features or indexing the aspects.

In this thesis I address some of these issues, building on the basis of former results by prof. Richard E Blake within feature extraction and graph matching [18, 19, 20, 22, 23].

The main objectives for this work are threefold:

- Develop an approach for obtaining an approximation of the aspect/appearance graph representation of arbitrary objects.
- Demonstrate a practical use of graph matching. Graph matching is often considered inefficient for practical applications due to its complexity, however using proper heuristics will lead to quick convergence in many cases.
- Present an application example for the aspect/appearance graph model.

That is, the goal is not to compute the exact aspect graph of the object, but to provide an approximation of it which is practical both in terms of size and required computation, still it has enough detail that it is useful for recognition and pose estimation purposes. Hence, the approach is as follows:

1. Simulate the appearance of the object over a set of viewpoints, based on a model. It is assumed that the model is descriptive enough to provide all the necessary information for the simulation of sensory data, for example it contains the model geometry and surface properties needed for generating RGB images using ray tracing.
2. Group similar views together. Similarity measure is determined via graph matching, where attributed relational graphs describing the object as seen from a given viewpoint are composed from both high and low level features extracted from the simulated data.
3. Refine the resolution and generate more views as needed. This refinement will enable narrowing down on the boundaries between the areas of the viewing space corresponding to different aspects.

1.2 Major contributions

The most important contribution of this thesis is the development of a generic framework for estimating the aspect graph or the appearance graph of an arbitrary rigid object.

In addition, a simple recognition / pose estimation system based on this framework is presented, including pose estimation tests based on both simulated and real-world data.

1.3 Outline of the thesis

The rest of the thesis is organised as follows. Chapter 2 gives a brief summary of recent research in computer vision, focusing on object modelling, object recognition and pose estimation.

The details of the modelling approach and the actual implementation are presented in Chapter 3.

Chapter 4 shows experimental results using both simulated and real data.

Finally, Chapter 5 gives a summary of the work, together with an evaluation of the results and implications for further work.

Chapter 2

Literature Survey

Computer vision, and in particular object recognition and pose estimation has been an area of intensive research in the past few decades, with a lot of published papers and numerous textbooks [9, 43, 76, 84, 87, 109]. Systems for automatising tasks accomplished by human (or other biological) vision [99, 71] are being developed and applied in various areas, from industrial manufacturing and inspection to surveillance and even tasks of such complexity as to guide a vehicle along a highway. In this chapter I present a survey of some of the underlying results.

2.1 Modelling

2.1.1 2D models

2D models are used for description or recognition of 2D scenes, such as line drawings. These models can also be useful in limited 3D recognition, where the viewing angle is given, or the 3D object has a limited number of possible viewpoints or stable configurations, and thus the 2D projection of the object can be predicted in advance. Examples of such applications are industrial part recognition or interpretation of aerial or satellite photography.

Jagadish and Gorman [86] has developed a line based image model (Thin Line Code, TLC) for 2D images, designed for image recognition applications. A line based recognition system for engineering drawings is presented by Blake [18]

Many of the techniques used for 2D recognition are also incorporated in methods for recognising 3D objects from 2D images.

2.1.2 $2\frac{1}{2}$D models

$2\frac{1}{2}$D models are 2D models extended with 3D object features, such as surface curvature or planar coefficients. These models can be considered as an intermediate step in scene interpretation, before an object centred model of the scene is constructed. $2\frac{1}{2}$D models can be obtained by extracting depth information with the help of range sensors or by projecting a known pattern on the image (Asada et al. [6, 5], Stockman et al. [154]). Tactile sensors can also be used to obtain depth data (Allen [2]). Another example for $2\frac{1}{2}$D primitives is object wings introduced by Chen and Stockman [39, 40, 153]. Object wings are composed of two adjacent surface patches and a separating contour segment.

2.1.3 3D models

The desirable properties for a solid modelling scheme has been established by Requicha [129]:

- the domain of the representation should be large enough to be able to represent a wide range of objects,
- the representation should be unambiguous, meaning that a particular representation can only represent one object,
- it should be unique, that is only one possible representation exists for each object, and
- it should be accurate.

Many different solid modelling techniques have been developed in the last decades for various domains including computer aided design and manufacturing, computer vision and visualisation, just to name a few. The characteristics of these techniques are quite diverse, so not all of them may be equally suitable for a particular type of application. The following sections summarise some of the most commonly used 3D modelling techniques.

Constructive solid geometry

Constructive solid geometry (CSG) applies regularised Boolean set operators to combine simple object primitives. The primitives are generic, and their placement, orientation and scaling

is selected before they are instantiated. CSG representations are not unique in the sense that a solid can be represented in several different ways using the same set of primitives. Since the CSG primitives are volumetric, boundary information must be computed, which imposes some difficulty on using these models in computer vision applications, particularly in edge based recognition systems.

Boundary representations

Boundary representations (BREPs) represent the object in terms of its surface boundaries, vertices, edges and faces. The modelled faces may be limited to polygons or triangles, or allow various types of curved surfaces. Many boundary representations limit the range of solids to 2-manifolds.

BREPs are often used in computer vision, as it is the surface boundary of a solid which is directly observable by common sensing devices (cameras, range sensors, tactile sensors, etc.). In addition, BREPs have the advantage of being unique, containing explicit information about volume, surface, edges and vertices, and having the flexibility for describing a wide range of objects [129, 14].

Wireframe models

Wireframe models consist of straight line segments which correspond to the edges of the solid. From the definition it is obvious that only polyhedral objects can be represented, objects with curved surfaces need to be approximated. As a result of the simplicity of this representation, wireframe models are easy to construct and to display. On the other hand, it is not trivial to verify if a wireframe model corresponds to a valid 3D object. Another drawback of this method is the ambiguity which arises when the model is projected to 2D; in order to solve this problem a hidden line removal algorithm can be applied. Wireframe models are often used in edge-based recognition systems, since the direct relation between the edges of the object and the model.

Sweep representations

The 3D object defined by a 2D area or 3D object moving along trajectory in 3D space is called a sweep representation. Commonly used special cases are moving a 2D area along a linear path (transitional sweep) or rotating it around an axis (rotational sweep). Generalised sweeps allow

the generating area or volume change in size, shape and orientation while being swept along an arbitrary curve. The generalised cylinders [17] in computer vision are generalised sweeps of a 2D cross section, and are usually modelled as a parametrised shape moved at right angles along a curve. Sweep representation by definition restricts the range of solids which can be modelled, but it becomes much more versatile when embedded in CSG systems.

Space subdivision

Voxel representation, the 3D equivalent of the 2D raster based on a uniform space subdivision, with equally sized voxels in a 3D grid. Octrees (the 3D extension of quadtrees) define a non uniform, hierarchical subdivision. Both representations suffer the difficulty of extracting boundary information. In addition, voxel representations require large amount of storage. Octrees improve on the storage requirements by not subdividing cells entirely on the inside or outside of the object. However, octree representations are difficult to compare as a slight change in the object or its orientation may result in a very different octree.

A more general approach is the binary space partition (BSP) tree, where the space is subdivided by planes of arbitrary orientation. Thibault and Naylor [157] present a method applying the BSP tree to represent arbitrary polyhedra. An important application of BSP trees is point classification (Tilove [158]).

Voxel and octree representations are commonly used in medical applications for visualisation and analysis of sensory data from various 3D scanning devices (such as MRI, CT or ultrasound). [173, 89]

2.1.4 Aspect graphs

The aspect graph of an object is a viewer-centred representation in form of a graph in which each node represents a topologically distinct view (*aspect*) of the object observed from some maximal connected area of viewpoint space, and each arc represents a transition from one region of viewpoint space to a neighbouring region (also called a *visual event*).

Aspect graphs have first been introduced by Koenderink and van Doorn [93] in 1979. Since then extensive research has been done in this area, and several algorithms have been developed for computing aspect graphs for different types of objects.

Polyhedral objects have received much of the attention (Castore [32, 33], Plantinga and Dyer [123, 124], Stewman and Bowyer [150, 151, 152], Watts [164], Bowyer and Dyer [28],

Kameyama and Nagata [88], Zha et.al. [176]). Gigus and Malik [67, 68, 69] and Gigus, Canny and Seidel [65, 66] discuss the computation of the aspect graph for line drawings of polyhedral objects under orthographic projection. Their definition of the aspect is based on the topological structure of the line drawing of the object, represented as an image structure graph (ISG), which can be substantially different from the definition by the visible faces of the object.

Laurentini [96] debates the usability of the ISG model for topological matching and proposes purely topological definition which does not suffer from similar problems.

Methods for computing the orthographic and perspective projection aspect graphs for solids of revolution have been proposed by Kriegman and Ponce [94, 95], Eggert and Bowyer [50, 54, 55, 51].

Chen and Freeman [37, 38] discuss the characteristic views of objects bounded by quadric surfaces. The aspect graph of solids with algebraic surfaces is studied by Ponce and Kriegman [125], Petitjean et.al. [121], Roy and Van Effelterre [132].

For arbitrary piecewise smooth opaque objects, Sripradisvarakul and Jain [144] present an algorithm for computing the orthographic projection aspect graph based on a strategy computing all accidental viewing directions that partition the viewing sphere from the shape descriptions of an object in a CAD database.

Van Effelterre, Van Gool and Oosterlinck [49] apply real algebraic geometry to compute the exact aspect graph of general CAD objects under perspective projection.

A medical image analysis application of the aspect graph is presented by Noble, Wilson and Ponce [112] addressing the problem of computing the orthographic projection aspect graph of smooth shapes from volumetric data.

A related viewer-centred approach for modelling the geometry of the occluding contour of a polyhedron, the rim appearance representation is presented by Seales and Dyer [136].

Weiss and Nawab [165] combine the aspect graph with the appearance model of the object into a representation denoted as appearance graph. They also present a method of searching the appearance graph at different levels of abstraction.

The aspect graph model has been generalised and extended to include objects with moving parts (articulated assemblies and objects with translational connections) by Sallam et.al. [134] and Bowyer et al. [27]. Kinoshita, Mutoh and Tanie [91, 92] describe the haptic aspect graph of 3D objects, using a high definition tactile sensor.

Faugeras et.al. [58] discuss a number of issues regarding the aspect graph representation. Some of the most important concerns are:

- the number of aspect increases very fast with the object complexity, and the aspect graph model does not say anything about their importance
- indexing of different views belonging to the aspects may be problematic
- does not consider real image features or feature recovery
- geometric information may not be enough
- computationally complex, graph is large
- extraction of line drawings is difficult from noisy images
- mixing geometric and brightness contours

In order to deal with the large number of aspects, whereas many representing visual detail which may never be seen by an observer in practise, Eggert et.al. [52, 53] introduced the scale space aspect graph. Pae and Ponce [115] present an approach for constructing the scale space aspect graph for solids of revolution.

Shimishoni and Ponce [140, 141] present an algorithm for computing the orthographic projection aspect graph of polyhedral objects observed by a camera of finite spatial resolution. They show that under these circumstances, theoretically different views may become equivalent and 'accidental' views may occur over finite areas of the viewing space.

Roy et al. [133] study the problems related to the construction of approximate aspect graphs (AAG) from noisy sensor data, and present an algorithm for constructing AAGs using an approach based on uniform partitioning. Seibert and Waxman [137] build aspect graph like object representations from a sequence of images, by tracking the appearance of the object over the sequence. The model is built up incrementally by matching new images to previously established aspects.

Aspect graph models have been applied to various problems in computer vision research. Bowyer et al. [29] present several results including a simple object recognition system based on aspect graphs. Ponce and Kriegman [126] use the exact aspect graph representation in predicting and interpreting line drawings of 3D curved objects. Pampagnin and Devy [116] present an object identification system capable of recognising a 3D object from a single image, where the object is given by a region-edge-vertex model and an aspect graph. Stark et al. [148, 149] apply aspect graphs for choosing the starting point and providing and guiding strategy for nonlinear optimisation based object recognition. This specific approach is also used in the ERRORS-2 recognition system by Bowyer et al. [26]. Dickinson, Pentland and Rosenfeld [46, 47] decompose the aspect representation to a limited set of aspects of 3D ob-

ject parts in a finite dictionary. Cyr and Kimia [42] apply a shape similarity metric to group similar views of an object into aspects.

Methods for motion tracking of objects using aspect graphs are presented by Dickinson et. al. [45], Ravela et al. [128].

2.2 Recognition and pose estimation

2.2.1 Model based recognition

Many different approaches have been developed for model based recognition. Some of the most popular paradigms are:

- constrained search: the search space is systematically reduced by testing predicates
- automated programming: decision rules and procedures are automatically built from model data
- evidence based techniques: evidence values (weights) are assigned to observed features and object identification is based on accumulated evidence
- geometric hashing: an invariant quantity derived from the features is used for indexing a hash table of identity/pose
- local feature set extraction: hypothesis of the identity and/or pose is obtained from a small set of features, and the rest of the features are used for verification

An excellent survey of model based recognition is presented by Chin and Dyer in [41].

The hypothesis verification paradigm is exploited by Wong et al. [170] for recognition of polyhedral objects.

The constrained search method is implemented in the BONSAI recognition system by Flynn and Jain [61].

Wallack and Canny [162] present a new data structure for indexing model features, called a tree grid, which provides a more compact representation than traditional hashing, and preserves spatial ordering in addition.

Stahs and Wahl [145] combine hashing techniques and hypothesis generation/verification for object recognition and pose estimation using a 3D robot sensor producing range images of

the scene.

A combination of indexing and hypothesis verification based on local feature sets is used by Beis and Lowe [11]. In their approach a learning indexing function is used to select best candidates given a set of features, and a final verification stage guarantees the reliability of the recognition. A similar approach is described by Stanchev and Vutov [146], in which a data-driven indexing scheme is used for hypothesis generation, while the verification is based on a standard model driven technique.

The ACRONYM system by Brooks [30] compiles recognition instructions for the interpretation algorithm from predictions of image features based on the model.

Forsyth et al. [63] study the applicability of invariant shape descriptors (which are unaffected by the orientation of the object) in 3D recognition.

Global descriptors based on probability distributions of shape features is used by Osada et al. [114]. The matching process is thus reduced to sampling the unknown object, normalisation and comparing the results to the probability distributions of the models.

Murase, Nayar and Nene[103, 110, 104] present an approach for recognising objects from visual appearance, where the appearance model (called *parametric eigenspace representation*) is built from a large number of views obtained automatically under varying viewing and illumination conditions. This model is also the basis of the Real-Time 100 recognition system [111], which is capable of recognising 100 complex 3-D objects. Sengel et al. [138] combines the parametric eigenspace model with statistical moments of image signatures. Automatic construction of object models from images is also exploited by Dorai, Wang, Jain and Mercer [48] (using surface depth data), Sullivan and Ponce [155, 156] (fitting triangular splines to silhouette data), Park and Subbaro [117] (tangent plane matching), Ulupinar and Nevatia [160] (observations on symmetries in figures).

2.2.2 CAD based recognition

Bhanu, Ho, Hansen and Henderson [15, 16, 73] make a connection between CAGD (Computer Aided Geometric Design) models and vision systems. Recognition strategies are generated automatically based on 3D geometric properties.

Arman and Aggarwal [4] use CAD models to compile model features into a recognition tree, which is used for systematically filtering out feature sets obtained from the segmented scene during the recognition phase.

Hoffman et al. [80, 79] experiment with fully automatic CAD driven object recognition using different types of sensors (intensity and range images).

The 3DPO system by Bolles et al. [25, 24] uses CAD models and a hypothesis-verification based strategy for estimating the position and orientation of 3D parts from range images.

2.2.3 Pose estimation based on point sets

Several techniques for pose estimation based on corresponding point sets are presented by Haralick et al. [75, 74], DeMenthon and Davis [44], Wenli and Lihua [166], Quan and Lan [127], Rosin [131], Hillenbrand and Hirzinger [78], Ansar and Daniilidis [3]. In addition to points, Hanek, Navab and Appel [72] use line and cylindrical features (often present in industrial environments) for estimating the camera position. Chang and Tsai [34] too use test functions based on line features for hypothesis verification. The pose estimation and tracking system by Yoon, DeSouza and Kak [174] uses circular-shape features instead of lines.

Nomura et al. [113, 113] use a fusion of shading and edge data in a pose estimation method which is considerably more tolerant to sensing errors than pure geometry based techniques. The method presented by Chen and Stockman [36, 35] is based on matching both object silhouettes and internal edgemaps. Brightness information is also used in shape-from-shading based approaches (e.g. Worthington and Hancock [171]).

Color information also proved to be useful for pose estimation and recognition tasks (Ekvall, Hoffman and Kragic [56], Slater and Healey [142]).

2.2.4 Neural networks

Wright and Fallside [172] combine CAD models with a back propagation neural network for estimating the object pose from wireframe images. Neural networks have been applied to pose estimation and object recognition problems by Hogg et al. [81] (synergetic networks), Khotanzad and Liou [90] (multilayer perceptron), Hati and Sengupta [77] (two-stage network), Nakano and Watanabe [108] (multistage networks, where also aspect graphs are used in one of the stages).

2.2.5 Genetic algorithm

Toyama, Shoji and Miyamichi [159] propose a model-based pose estimation method based on a genetic algorithm with the viewing coordinates and rotation angles encoded in the chromosome structure. They apply a fitness method using edge direction for evaluating candidates.

Bergevin and Levine [13] address the problem of recognising generic objects from single 2-D images. They present a vision system which is not based on accurate object model, but coarse description of object classes. The generic system developed by Stark and Bowyer [147] uses an indexing scheme to recognise objects based on functional properties.

2.2.6 Structural models and relational descriptions

Matching structural descriptions represented by graphs is considered as one of the most difficult problems in computer vision. Complexity of exact graph matching in general remains to be classified, however several special cases have proven to be NP-complete (Basin [10], Garey and Johnson [64]). Inexact graph matching has also been proven to be NP-complete (Abdulkader [1]). For some particular types of graphs the problem becomes polynomial, as for planar graphs (Hopcroft and Wong [82]) or trees and forests (Matula [101], Reyner [130]), but in the general case heuristics are required to be able to treat the problem in practise.

Luo and Hancock [98] treat inexact graph matching in the purely structural sense as a maximum likelihood problem, and present an efficient matching algorithm using on this framework. Blake [22] proposes partitioning the problem into sub-problems based on a lattice of constraints ([20, 21]). Messmer and Bunke [102] has developed an approach for error correcting matching of an unknown graph to a set of model graphs, where the model graphs are preprocessed and compiled into a compact representation. The influence of a cost function on error correcting graph matching is studied by Bunke [31]. Van Wyk, Durrani and van Wyk [161] apply a reproducing kernel Hilbert space interpolator based algorithm to various graph matching problem and demonstrate the efficiency of this approach. An optimisation technique using graduated assignment is introduced for graph matching by Gold and Rangarajan [70], with $O(lm)$ complexity (where l, m are the number of links in the two graphs).

Shapiro and Haralick [139] propose a metric for comparing relational descriptions based on comparing weighted primitives (weighted attributes and weighted relation tuples) using a normalised distance for each primitive property that is inexactly matched.

Wilson and Hancock [167, 168] describe a Bayesian framework for matching relational graphs,

and a comparative study of different strategies for quantifying inexactness. According to their findings, graph editing techniques show the best performance. Edit distance for graph matching was introduced by Sanfeliu and Fu [135], and it is discussed in several papers (Myers, Wilson and Hancock [105, 106, 107], Petrakis, Faloutsos and Lin [122]). Llados, Marti and Villanueva [97] propose an error-tolerant algorithm for subgraph-isomorphism based on graph editing. Finch et al. [59] present a mean field annealing based technique using the Bayesian model.

Backer and Gebrands [7] apply inexact graph matching to structure graphs representing stereo images, with the graph nodes corresponding to object vertices.

The structural descriptions used by Bennamoun and Boashash [12] represents the object in terms decomposing it to convex parts obtained from the contour and modelled by superquadrics.

Fan, Medioni and Nevatia [57] describe a scene using relational graphs. Nodes of the graph correspond to visible surfaces patches obtained from dense range images and edges describe the relations between them. Recognition using graph matching is based on a set of descriptions from different view angles.

A hierarchical graph representation of objects is proposed by De Floriani [60], based on connectivity properties of the generalised edge-face graph (GEFG) and form features.

Flynn and Jain [62] describe a system for generating relational graph descriptions of objects from CAD models, composed of both view-independent elements from IGES descriptions and and view-dependent elements based on synthetic views.

Wong, Lu and Rioux [169] propose an object description scheme using attributed hypergraph representation (AHR). Nodes of the AHR correspond to primitive blocks of the object which too are represented by attributed graphs. AHR's are constructed from single range images, then combined to the so called complete AHR which is an orientation invariant description of the object.

An indexing technique for models based on graph representation, graph hashing is presented by Sossa and Horaud [143].

Chapter 3

Approach & Implementation

3.1 Overview of the modelling system

In this chapter we present the approach and implementation details of the modelling system based on an approximation of the aspect/appearance graph of objects. The purpose of the system is to convert the geometric model of an object, extended with additional information such as surface properties and environmental (e.g. lighting) conditions, to a representation which can be used in a computer vision system for pose estimation or object recognition.

An overview of the modelling system is presented in Figure 3.1. The geometric model of the object is used as a starting point for the modelling. This model is extended with extra information, related to the object (e.g. surface properties) or to the environment (e.g. lighting conditions). The model building is based on the tessellation of a viewing sphere. The object is placed in the centre of the sphere, and the views from different locations on the surface are analysed. Ray tracing is used for image generation. The images are segmented, and a set of high level features are extracted for each image region. Graph matching is applied to compare adjacent views. The surface patches are then subdivided and the procedure is repeated until the match between the adjacent views is good enough or a desired resolution is reached.

From the information gathered during the viewing sphere tessellation, equivalence classes of views are extracted. This is done by merging the patches of the viewing sphere. Depending on the model parameters and environmental conditions, these equivalence classes can approximate the aspect and/or appearance graph of the object.

In order to give a description of the classes for use in recognition / pose estimation, contour types are introduced. Classes are represented as sets of contours where the elements are disjunctions of contour types.

3.2 Use of ray tracing for model building

In model based object recognition, the aim is to find the relation between a description of an object and the data we get from the acquisition device or devices. Different types of acquisition devices perceive the same object in different ways. A camera gives a 2-D colour or grey-scale image of the object seen from a given viewpoint, a range scanner measures the distance to the surface points, a thermal camera would give a map of thermal variations. The data received from the acquisition device is preprocessed, and set of features which characterise the objects are extracted. The observable features vary from one acquisition device to the other, commonly used features include (but are not limited to) edges, corners, surface properties (e.g. colour, orientation). It is not always obvious from an object model –for example CAD models, or constructive solid geometry models–, what object features can be detected by some acquisition device from a given viewpoint. In order to fill this gap, we need to do some preprocessing of the object model.

The approach which is presented here is based on simulating the behaviour of the acquisition device. The simulated data from a virtual acquisition device is processed by the same feature extraction tools as the real image data. The extracted features will therefore have the same properties as if they were extracted from real images, so they can be used for evaluating the similarity between simulated and real data.

In the following, we will concentrate on an RGB camera as an acquisition device, since it was used for conducting the experiments in Chapter 4. Nevertheless, no limitations are imposed on the modelling system by this choice, only the set of observable features is influenced.

3.2.1 From geometric models to images

The interesting features for the RGB camera (and many other acquisition devices) are the object boundaries –surfaces, edges and corners. Some geometric models describe the object by defining its boundaries (BREP), others define the volume itself (CSG, voxels). The latter needs to go through extra processing to produce the features observable for a camera.

The object models in this work are based on the simple boundary representation used by the

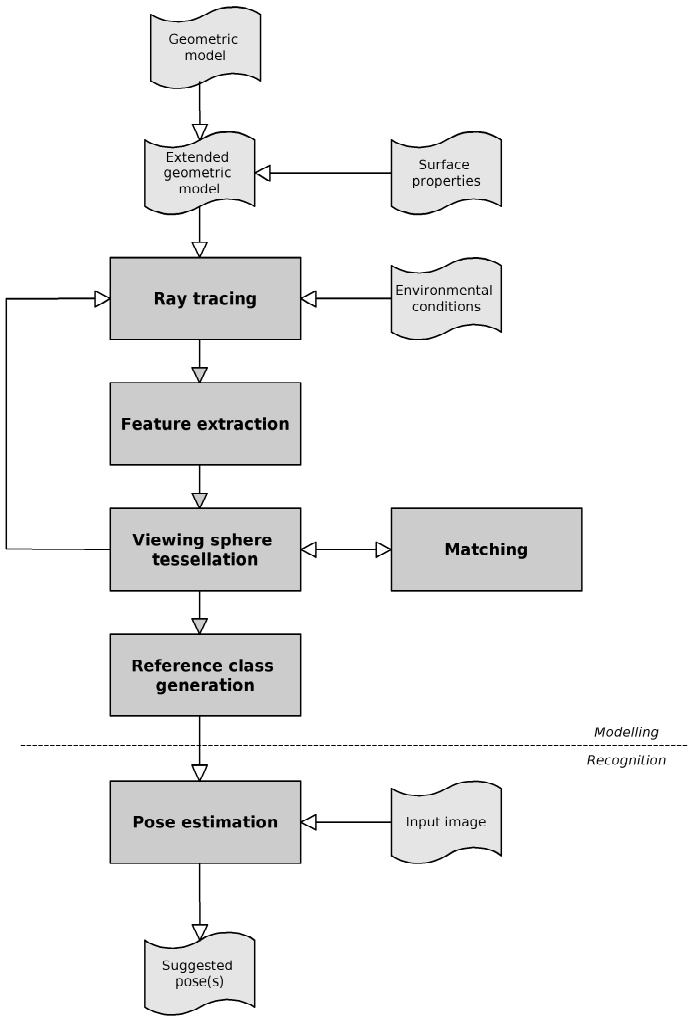
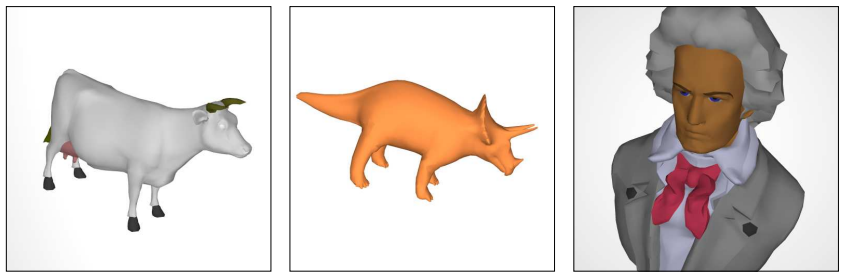


Figure 3.1: Overview of the modelling system

Bob ray tracer [163]. This representation supports polygonal faces, circular, spherical and conic surfaces. Other, more complex surfaces are approximated with a polygonal (e.g. triangular) tessellation (Figure 3.2). A conversion tool from AutoCAD DXF files has also been developed.



(a) (b) (c)

Figure 3.2: Ray traced surfaces approximated by triangles

For the simulation of the images acquired by a camera, the geometric model has to be augmented with additional data about the surfaces of the object. Each surface may have a number of different properties, depending on the method for generating images from the model. For ray tracing –the method used by the modelling system– these are the following:

- diffuse reflection (given by colour components r:g:b)
- specular reflection (r:g:b)
- transparency
- index of refraction
- optional surface texture

The decision to use ray tracing as a rendering method is explained by its flexibility while still being relatively fast. However, ray tracing has its shortcomings too, and it is not unthinkable that other methods produce better results in different environments. With the extensions of simulating ambient light and soft shadows, ray tracing did perform well within the setting of this thesis. Other types of sensors (range scanner, thermal camera) require a corresponding simulation method. It is important to keep in mind that the modelling system is based on generating a large number of views, so execution speed is an important factor to consider when deciding on a method.

3.2.2 High level features

The high level features used in model building and recognition are collected in attributed relational graphs, which are commonly used in model based recognition [23]. The extraction of these features from images –whether they contain real image data or simulated data– is done by the same pipeline of image processing tools. The image is segmented into connected regions based on the uniformity of the colour ratio, r:g:b, while the intensity is above a given threshold, and the required connectivity criterion is 4-connectedness [118]. The contours of the image regions are traced, and high level features describing the contours and the relations between the contours are computed. Finally all of these features are collected in an attributed relational graph representing the image.

Image segmentation

Prior to image segmentation itself, some image enhancements are performed to improve the image quality and thus make the segmentation more robust. These enhancement steps are obviously more important for real data than for simulated images, nonetheless the pipeline is executed the same way regardless of the source of the input.

The major steps of the image enhancement and segmentation are as follows:

1. smooth of the image without smearing edges
2. normalise colour values and reduce variance while keeping high variance boundaries between regions
3. segment the image into regions based on colour separation, using 4-connectedness as the connectivity criterion and the uniformity of the r:g:b colour ratio with total intensity over a threshold as the common property (as in the standard definition of a connected region[1])
4. discard very small regions and filaments
5. trace the contour of each region, and compute the contour attributes: mean colour components, centre of gravity, area, moments, shape classification

[1] Set of pixels that are connected under some definition and share a common property.

Contours and attributes: Nodes of the relational graphs

From the contours of each image region, a list of attributes is computed. Currently, this list consists of the following:

Contour type	ext/int: external or internal contour flag
Shape	tri/sqr/cir/unk: simple shape classification: triangular, square, circular or unknown
Area	aa(area): the area enclosed by the contour
Moments	mm(I_1,I_2,I_3,I_4): the first four rotation invariant combinations of normalised central moments [85] $I_1 = \eta_{20} + \eta_{02}$ $I_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2$ $I_3 = (\eta_{30} - 3\eta_{12})^2 + (\eta_{03} - 3\eta_{21})^2$ $I_3 = (\eta_{30} + \eta_{12})^2 + (\eta_{03} + \eta_{21})^2$
Centre	cg(x,y): coordinates of the centroid
Colour	cr(r,g,b): mean colour values around the track of the contour
Planar	pl(A,B,C,D): plane equation coefficients for the best fitting plane (if range image data is available) $Ax + By + Cz + D = 0$

Each contour is assigned a *contour id* (**A**, **B**, ...). The contour id is not related to the contour attributes, but to the order in which contours are found in the image. A different orientation of the image may therefore result in a different assignment of contour id's (this problem will be addressed later).

Relations between contours: Arcs of the relational graph

The edge attributes are computed from the spatial relation between the centres of the contours. The relation is encoded in a 16-way direction code (Figure 3.3). The code letters s, i and o are used in addition to specify that one contour is superimposed, laying inside or outside an other.

In order to reduce the size of the graph, the number of arcs included in the graph can be limited. An easy way to do this is is to keep only the first n arcs sorted by the distance between the centroids of the contours they connect. Using a function of the area or other contour parameters as an importance factor is also possible, thus making connections between the

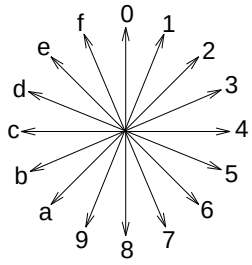


Figure 3.3: 16-way direction codes

contours of important (e.g. large) regions more likely to be considered. It is usually a good idea to include enough arcs to keep the graph connected (although not required for the graph matching). This could be achieved by increasing the maximum number of arcs allowed if necessary, until the graph becomes connected. Another possibility is to find a spanning tree (e.g. minimal spanning tree) of the graph, and then include additional edges by the above criteria.

Figures 3.4 and 3.5 show an example segmented image with the graph drawn on top, and the textual representation of the relational graph including the node and edge attributes.

3.3 Viewing space tessellation

Concerning the tessellation of the viewing space, a number of problems must be addressed. One of them, and perhaps the most important is what resolution can be achieved. The optimal resolution depends on several factors. Choosing too low resolution results in a loss of detail, while too high resolution means extra computation time which would otherwise be unnecessary. On the modelling side, the image resolution could theoretically be increased to infinity, while on the input side the acquisition devices have their hard limits. In practise, the usable image size can be even smaller, depending on the computation costs involved in preprocessing, feature extraction, and model generation.

There are several different ways of tessellating a spherical surface. Meshes based on cylindrical or spherical coordinates are easy to compute, finding adjacent patches is trivial, and the resolution can be increased arbitrarily. The drawback of these methods is the variation of the shape and size of the patches over different areas of the sphere. Using regular polyhedra or

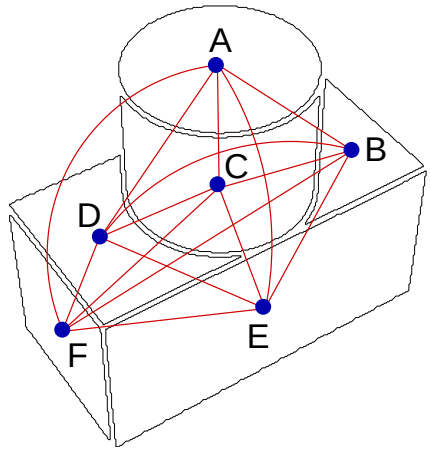


Figure 3.4: Segmented image and the corresponding relational graph

triangular symmetry groups [8] gives a uniform distribution of patches over the sphere, but the number of these tessellations is limited (there are only five convex regular polyhedra[2] and four types of triangle groups[3]), and none of these provides an acceptable resolution.

3.3.1 Icosahedron method

This tessellation method is based on an icosahedron. The regular icosahedron ('level 0 icosahedron') has 20 triangular faces.

The basic idea is to divide the faces of the solid into smaller facets. The triangular faces are split into four triangles, using the mid-points of the edges as new corners. Each new triangle thus covers one fourth of the area of the original. The new corners are then projected back to the surface of the sphere. While the triangles we get from the subdivision are, except for scaling and rotation, exact copies of the original, the projection of the corners distorts them so

[2]The regular polyhedra (*Platonic solids*) are the tetrahedron(4,4,6), cube(6,8,12), octahedron(8,6,12), dodecahedron(12,20,30) and icosahedron(20,12,30). The number of faces, vertices and edges is indicated in parentheses.

[3]The valid configurations are $(2, 2, n)$, $(2, 3, 3)$, $(2, 3, 4)$ and $(2, 3, 5)$, where the numbers indicate the angles of the spherical triangles composing the symmetry group: $(\frac{\pi}{p}, \frac{\pi}{q}, \frac{\pi}{r})$

A ext:cir:aa(9941):mm(21243,43,41,1):cg(255,334):cr(255,167,139)
A–[6]→B A–[8]→C A–[9]→D A–[8]→E A–[9]→F
B ext:tri:aa(3772):mm(15848,10,278828,1144):cg(345,273):cr(210,245,227)
B–[c]→C B–[b]→D B–[9]→E B–[b]→F B–[e]→A
C ext:tri:aa(10537):mm(16538,29,213276,1329):cg(255,264):cr(46,53,87)
C–[a]→D C–[7]→E C–[a]→F C–[0]→A C–[4]→B
D ext:tri:aa(7703):mm(14718,41,53949,1221):cg(187,214):cr(161,189,175)
D–[5]→E D–[a]→F D–[1]→A D–[3]→B D–[2]→C
E ext:sqr:aa(17692):mm(18114,194,9471,241):cg(286,169):cr(196,130,119)
E–[c]→F E–[0]→A E–[1]→B E–[f]→C E–[d]→D
F ext:sqr:aa(4227):mm(18898,266,1343,213):cg(145,155):cr(33,38,62)
F–[1]→A F–[3]→B F–[2]→C F–[2]→D F–[4]→E

Figure 3.5: Textual representation of the relational graph for the image on Figure 3.4

they are no longer exactly the same size and shape. For practical reasons, this is no problem, as the difference is not too large (the relative difference from the average area is within $\pm 15\%$ even after seven iterations).

The solids, which are generated this way are called 'level n icosahedrons', where n indicates the number of iterations, i.e. how many times were the faces divided. Some examples are shown on Figure 3.6.

For simplicity, all object models were defined (translated and scaled if necessary) in such way that the object was centred at the origin, and the viewing sphere is the unit sphere (the *Gaussian sphere*). The equations in (3.1) give the corners of the new patches under these conditions (the corner vectors of the original patch were $(\vec{r}_1, \vec{r}_2, \vec{r}_3)$).

$$
\begin{array}{lcllcl}
\vec{s}_{11} &=& \vec{r}_1 & \vec{s}_{31} &=& \vec{s}_{13} \\
\vec{s}_{12} &=& \frac{\vec{r}_1+\vec{r}_2}{\|\vec{r}_1+\vec{r}_2\|} & \vec{s}_{32} &=& \vec{s}_{23} \\
\vec{s}_{13} &=& \frac{\vec{r}_1+\vec{r}_3}{\|\vec{r}_1+\vec{r}_3\|} & \vec{s}_{33} &=& \vec{r}_3 \\
 & & & & & \\
\vec{s}_{21} &=& \vec{s}_{12} & \vec{s}_{41} &=& \vec{s}_{23} \\
\vec{s}_{22} &=& \vec{r}_2 & \vec{s}_{42} &=& \vec{s}_{13} \\
\vec{s}_{23} &=& \frac{\vec{r}_2+\vec{r}_3}{\|\vec{r}_2+\vec{r}_3\|} & \vec{s}_{43} &=& \vec{s}_{12}
\end{array}
\qquad (3.1)
$$

As the ultimate goal of the viewing sphere tessellation is to end up with connected regions of patches which represent the different aspects of the objects, we need to keep track of which

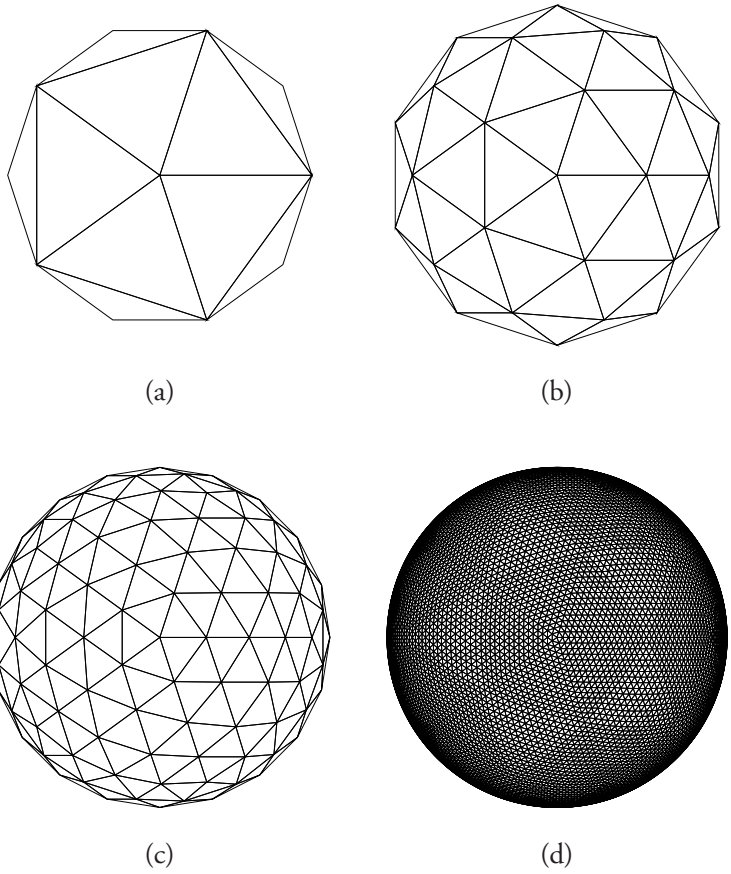


Figure 3.6: (a) Regular icosahedron, 20 faces; (b) level one icosahedron, 80 faces; (c) level 2 icosahedron, 320 faces; (d) level 5 icosahedron, 20480 faces

patches are adjacent to each other. This is relatively easy as long as the patches are all the same size, but it soon becomes more complicated when patches of different sizes are present – which is almost always the case, as the subdivision is done in an adaptive manner (see Section 3.3.2). For this purpose, a patch adjacency graph is built for the level-0 icosahedron, and it is continuously updated each time a patch is divided. The adjacency graph is in fact represented as a directed graph where edges for adjacent patches are added both ways. The reason for this is that the ordering of the outgoing edges for a node is crucial for the graph update procedure during patch subdivision.

To understand how the updates are performed, let us consider a patch P, with vertices v_1, v_2 and v_3, which will be divided to four new sub-patches P_1, P_2, P_3 and P_4 according to (3.1). The list Q_1, Q_2, ..., Q_n are the neighbours of P_1, starting from vertex v_1 along the side (v_1, v_2), then further on along (v_2, v_3) and finally (v_3, v_1). When the subdivision is performed, the new edges are introduced to the graph in such a manner that this property is preserved for both the old and new patches.

Let us start with patch P_1. P and P_1 share the corner v_1, which means that the fist neighbour of P_1 will be Q_1 (which was the first neighbour of P). Now, consider the size of Q_1 compared to P (or rather the level of subdivision at which they were added, $l(P)$ and $l(Q_1)$).

If $l(Q_1) \leq l(P)$ (Q_1 is the same size or larger than P), that means that Q_1 will be the only neighbour of P_1 on the first side, and Q_1 will be the fist neighbour of P_2 as well (and the only one at P_2's first side). We therefore remove the edge (Q_1, P), and replace it with (Q_1, P_2) and (Q_1, P_1), at the same place on Q_1's adjacency list where (Q_1, P) was located (note that the order of $P1$ and $P2$ is reversed).

On the other hand if $l(Q_1) > l(P)$ (Q_1 is smaller than P), that means that P_1 will have one or more neighbours along its first side, none of which are shared with P_2. To determine how many of P's neighbours will P_1 get, we look at the subdivision levels of these. We know that that side of a level (n+1) patch is half the length[4] of the side a level n patch. This means that we need to find k neighbours $Q_1, ..., Q_k$ such that

$$\sum_{i=1}^{k} 2^{-l(Q_i)} = 2^{-l(P_1)} \tag{3.2}$$

For the patches $Q_1, ..., Q_k$, P_1 will simply replace P on the adjacency list. At the same time, $Q_1, ..., Q_k$ is added to P_1's adjacency list.

[4]This is not quite correct as the patch vertices are projected back to the surface of the Gaussian sphere; still the argumentation based on patch levels is valid

The same procedure is then repeated for all three sides of patch P. In addition, the edges (P_1, P_4), (P_2, P_4) and (P_3, P_4) are added when appropriate (for example, (P_1, P_4) is added after (P_1, Q_k)). P_4 is only adjacent to P_1, P_2 and P_3, in this order. Finally, all edges to P have been removed / replaced by edges to the new patches, so P can safely be removed.

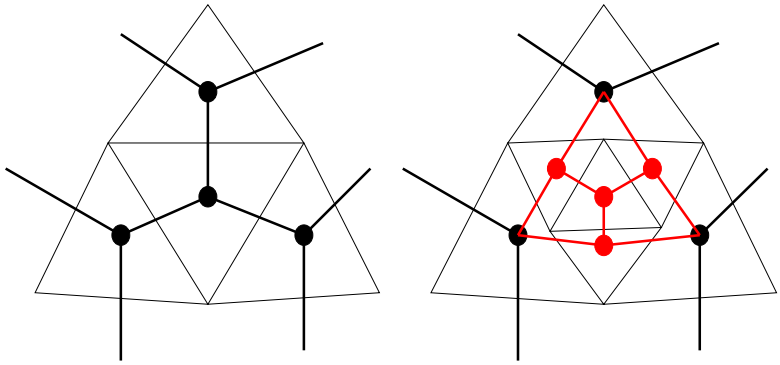


Figure 3.7: A segment of the graph of viewing sphere patches and neighbour relations before(a) and after(b) the triangle subdivision

3.3.2 Adaptive subdivision

The issue of finding the appropriate tessellation resolution for a specific object is addressed by adaptive subdivision. Patches which cover an area of the viewing sphere where the views show acceptable similarity do not need to be subdivided. On the other hand, if there are large differences between the views, the patch is subdivided.

In practise, the similarity check is done by applying graph matching [19] to the views from the corners of each patch. For each corner, a synthetic image of the object is generated and processed by the feature extraction tools, yielding a relational graph for that viewpoint. The similarity is measured via calculating the cost of pairwise matching these graphs to each other. A patch is accepted if all corner pairs have a match cost within a threshold, or subdivided otherwise.

The process is controlled by a number of parameters. First, a minimum and maximum subdivision level can be specified. The minimum level is used to enforce that a certain amount of detail is achieved; all patches will be subdivided until the minimum level is reached, regardless of the graph matching results (which are not even computed in this case). The maximum

level gives a stopping criterion for the subdivision: patches at this level will not be subdivided anymore (whether these patches are accepted or not still depends on the match costs).

A second set of parameters controls the graph matching itself. The relative importance and scale of the graph attributes can be adjusted by a set of weights, and the decision whether the match is good enough is controlled by a threshold. Transforming the results of the graph matching from a multi-dimensional parameter set to a single scalar cost (or distance) function is not a trivial problem, and well worth dedicated research. However, for the purposes of this thesis the graph matching tool has been applied as-is, and the cost functions will not be discussed in detail.

Another parameter, which can also serve as stopping criterion, is the total area covered by the accepted patches. As the subdivision level increases, and the patches become small enough to have good similarity between the corners, the unaccepted patches will be concentrated along boundaries between different areas of the viewing space which correspond to different aspects of the object. By increasing the subdivision level, these boundaries get narrower and the uncovered area is decreasing. In the ideal case, when all unaccepted patches are crossing the boundary, the uncovered area will be reduced by one half with each level increase (the patch size will be one fourth, but we need twice as many patches along the boundary). As an example, this tendency can be observed for the CUBE test object in Section 4.3.1.

The number of patches grows exponentially with the level of subdivision, which puts a practical limit on how far the subdivision level can be increased. Nevertheless, the resolution is also doubled for each level, so acceptable precision is reached after relatively few iterations. In the worst case, when every patch needs to be subdivided, the number of patches will be $t_l = 20 \cdot 4^l$, where l is the level of subdivision ($t_1 = 80$, $t_2 = 320$, $t_3 = 1280$, $t_4 = 5120$, $t_5 = 20480$, $t_6 = 81920$, $t_7 = 327680$, etc.). At the same time, the distance between adjacent views is halved: $s_l = s_0 \cdot 2^{-l}$. This means that the number of patches is quadratic function of the resolution $r_l = s_l^{-1}$, that is $t_l = C_1 \cdot r_l^2$. In the ideal case, when all patches which do not cross boundaries are accepted, the number of patches remaining is approximately doubled, so the patch count in this case will be closer to a linear function of the resolution $t'_l \approx C_2 \cdot r_l$. In practise the growth is quadratic in the beginning, and gets closer to linear after a few iterations, when the resolution is high enough to yield good matches between adjacent views (see detailed results in Chapter 4). Note that the number of aspects for the various objects are quite different, and so is the total length the boundaries between the aspects and the constant C_2.

3.4 Building reference classes

The result of the adaptive subdivision algorithm is a tessellation of the viewing sphere, represented by a graph where the nodes correspond to the triangular patches and the edges connect the adjacent patches. The next step is to cluster the patches in order to obtain the reference classes describing the different aspects of the object. The patch clusters are converted to clusters of views using the corners of the patches. Then the image features extracted from these views are used to build a class description for each cluster.

3.4.1 Clustering the patches

Clustering the patches (Algorithm 3.1) is performed in a region growing manner. The process starts with an arbitrary patch, and if it is uniform all neighbouring patches are collected which are also uniform and match the starting patch. Non-uniform patches are discarded. This step is repeated for the newly collected patches, until no more patches are added to the cluster. Then a new starting patch is picked for the next cluster, and so on until there are no patches left. An example for the result of the patch clustering is shown on Figure 3.8, where each cluster is assigned a random colour.

For each cluster, the list of patches is converted to a list of viewpoints using the corners of the patches. As several (up to six) patches share the same corner, duplicates from this list are removed. Optionally, a weight can be calculated for each viewpoint, which is defined by the sum of the area of the patches in the cluster which that viewpoint is a corner of, divided by three. These weight values give an indication of approximately how large area of the viewing sphere a particular view can be associated with. In addition, the sum of the weights for all views give the total area of the cluster.

3.4.2 Consistent naming of contours

For object recognition or pose estimation purposes a cluster of views is not the most efficient way of representing the object, especially for large clusters. It is therefore desired to have a simplified description of the classes combining the similarities of the views within the class and thus allowing for quick decisions to which class an unknown view belongs to. One possibility is to represent each class with one single view from a point at (or near) the cluster centre. This approach –although it may seem simple and efficient– has a number of problems. First, the shape of the clusters are often quite complex, so it is not always easy to define where the

Algorithm 3.1 Patch clustering

Input: L_1 contains a list of all patches
Output: PL contains a list of all clusters

```
PL ⇐ {}
while L1 ≠ {} do
  L2 ⇐ {}
  p ⇐ {}
  append(L2, pop(L1))
  while L2 ≠ {} do
    n ⇐ pop(L2)
    if uniform(n) then
      append(p, n)
      for all nn such that adjacent(nn, n) do
        if nn ∈ L1 ∧ uniform(nn) ∧ match(n, nn) then
          append(L2, nn)
          delete(L1, nn)
        end if
      end for
    end if
  end while
  if p ≠ {} then
    append(PL, p)
  end if
end while
```

centre of a cluster is. Second, even if we manage to find a good algorithm for locating the cluster centre (or we select it manually), a single view may not describe the whole class appropriately. These problems are addressed by collecting feature information from all views within a cluster, and merging it into a compact representation, which describes the common properties of the cluster. The clusters of views described this way will be referred to as *equivalence classes* in the following.

Most of the information about image features is carried by the node (contour) attributes. It is therefore not unreasonable to use the contour information as input for the description of the equivalence class. Earlier, each image contour has been assigned a contour id. It is however not guaranteed that two contours having the same contour id in different images correspond to the same surface of the object, since the contour id's are assigned in the order the contours are detected in the image. Even a small rotation of the image may be enough to change the

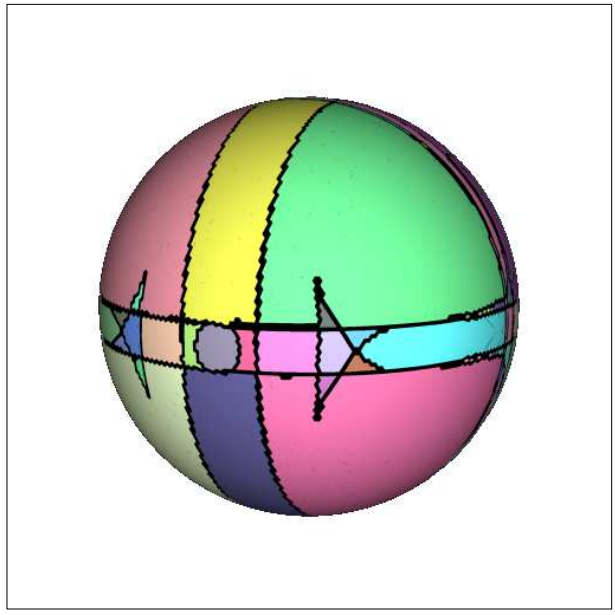
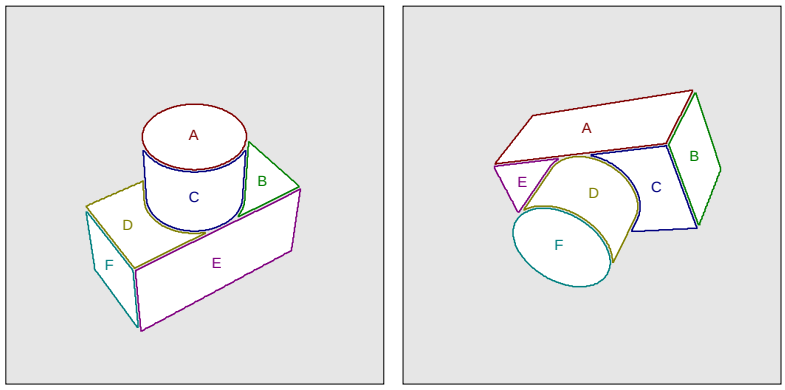


Figure 3.8: Result of patch clustering (BLOCKCYL object), with random colours assigned to each cluster

detection order (see Figure 3.9). For this reason, the first step is to establish a consistent naming of contours throughout the whole cluster. This is where the graph matching results from the adaptive subdivision (Section 3.3.2) will be used a second time. We know that all views within the cluster come from corners of accepted patches, i.e. the match between the corners was good enough. Graph matching, in addition to the cost of the match between two graphs, returns a list of node equivalences, that is a one-to-one mapping of the nodes of graph G_1 to the nodes of graph G_2 which is associated with the lowest match cost. This mapping can be used to systematically rename the contours of G_2, so the corresponding contours have the same contour id in both graphs. Table 3.1 shows an example, where the unknown and reference graphs come from the images on Figure 3.9.

A consistent naming of the contours for all views within the cluster is achieved by picking an arbitrary viewpoint from the cluster and remapping the contours of its neighbours according to the results from graph matching; then propagating the mapping to the neighbours of the neighbours, and so on until the whole cluster is processed.



(a) (b)

Figure 3.9: Renaming of the contours. (a)Reference view and a (b)view with different contour naming

ref idx	unk idx	cost	ref = unk
1	6	731	A = F
2	5	1121	B = E
3	4	1136	C = D
4	3	898	D = C
5	1	997	E = A
6	2	988	F = B

Table 3.1: Table of node equivalences and costs, from the output of graph matching

3.4.3 Contour types

Now that the consistent naming has been established, there is still a problem with the high variation of feature properties over the cluster. Some contours of the image will change in area and shape as we move along the viewing sphere, while others remain similar over large areas. Keeping all the different views in a cluster result in an enormous search space, and makes the pose estimation or recognition process inefficient. In order to reduce the size of the search space, *contour types* are introduced. Instead of keeping all details of all contours, we try to categorise the contours into a limited number of different types. The optimal number of the contour types depends on the complexity of the object, in general it should be large enough to cover the common variations of the contours which can be observed, but it shouldn't be too large either, since the whole idea is to reduce complexity. In practise, numbers between

30 and 100 give reasonably good results for the test objects, where the average number of contours is between 5 and 20 for a single image and 10^6 to 10^7 in total for all images.

The purpose of the contour types is to describe the similarities of a set of contours in a simplified manner. For example, we could say that one type is 'large red squares' and an other is 'small green circles'. However, such types are not easily defined automatically. A well established way to do such automatic categorisation is to apply an unsupervised clustering algorithm to the data. For the contour data, we use k-means clustering, applied to vectors of the appropriately scaled numerical contour attributes. Scaling was necessary as the range of the different components is quite different, e.g. RGB colour components are within 0-255, while the area can be up to 262144 pixels in theory (for a 512 by 512 image). The scaling parameters are determined from the mean and minimum/maximum values for each component, then adjusted to reflect relative importance or robustness of the components (e.g. colour is more important than area), similar to the graph matching parameters in Section 3.3.2.

The clustering is done on the parameter vectors of all external contours from all the views in all clusters. The centres of the resulting k clusters (the mean of all vectors in that cluster, $\vec{m}_i$) define k distinct contour types $\theta_1, \theta_2, \ldots, \theta_k$. A contour belongs to type θ_i, if the distance from its property vector to $\vec{m}_i$ is minimal.

3.4.4 Description of reference classes

Using the contour types, the equivalence classes for the views can now be described the following way:

For each cluster, label the contours for each view with the contour types. Then, using the now consistent contour id's, collect all possible types for contour **A**, then **B**, and so on. The result is a set of types for each contour id (meaning: "for an arbitrary view within this class, contour **X** can be one of these types"). A scoring scheme for the types is also possible, using one of three alternatives:

- No scoring: each type counts the same.
- Scoring by the number of views: count the number of views where a contour is of a particular type.
- Scoring by the weights of the views: use the weights assigned to the views during clustering (Section 3.4.1), the score is the sum of the weights of the views where the contour is of a particular type.

In the second and third case, the set of contour types is replaced by a set of pairs in the form

(contour type:score).

As an example, let us look at a class consisting of four different views with the following type assignments:

$$\begin{array}{rcl} graph_1 & = & \{A:\theta_4, B:\theta_{17}, C:\theta_9, D:\theta_{23}, E:\theta_{26}\} \\ graph_2 & = & \{A:\theta_{11}, B:\theta_{20}, C:\theta_4, D:\theta_6, E:\theta_{26}\} \\ graph_3 & = & \{A:\theta_9, B:\theta_{24}, C:\theta_9, D:\theta_{15}, E:\theta_2\} \\ graph_4 & = & \{A:\theta_4, B:\theta_{24}, C:\theta_9, D:\theta_{23}, E:\theta_{26}\} \end{array} \tag{3.3}$$

Furthermore, let us suppose that the views have the following weights: $w_1 = 0.3, w_2 = 0.1, w_3 = 0.1, w_4 = 0.2$. The description of the class using the first (no scoring) scheme will then be:

$$class_1^{(1)} = \left\{ \begin{array}{ll} A: & \{\theta_4, \theta_9, \theta_{11}\} \\ B: & \{\theta_{17}, \theta_{20}, \theta_{24}\} \\ C: & \{\theta_4, \theta_9\} \\ D: & \{\theta_6, \theta_{15}, \theta_{23}\} \\ E: & \{\theta_2, \theta_{26}\} \end{array} \right\} \tag{3.4}$$

The same class, using the second and third scoring schemes:

$$class_1^{(2)} = \left\{ \begin{array}{ll} A: & \{(\theta_4:2), (\theta_9:1), (\theta_{11}:1)\} \\ B: & \{(\theta_{17}:1), (\theta_{20}:1), (\theta_{24}:2)\} \\ C: & \{(\theta_4:1), (\theta_9:4)\} \\ D: & \{(\theta_6:1), (\theta_{15}:1), (\theta_{23}:2)\} \\ E: & \{(\theta_2:1), (\theta_{26}:3)\} \end{array} \right\} \tag{3.5}$$

$$class_1^{(3)} = \left\{ \begin{array}{ll} A: & \{(\theta_4:0.5), (\theta_9:0.1), (\theta_{11}:0.1)\} \\ B: & \{(\theta_{17}:0.3), (\theta_{20}:0.1), (\theta_{24}:0.3)\} \\ C: & \{(\theta_4:0.1), (\theta_9:0.6)\} \\ D: & \{(\theta_6:0.1), (\theta_{15}:0.1), (\theta_{23}:0.5)\} \\ E: & \{(\theta_2:0.1), (\theta_{26}:0.6)\} \end{array} \right\} \tag{3.6}$$

Table 3.2 shows a real life example of some equivalence classes of views of the BLOCKCYL object, using the second (number of views) scoring scheme.

$class_1$	A	$(\theta_{23} : 110)$
	B	$(\theta_7 : 1), (\theta_{22} : 42), (\theta_{25} : 67)$
$class_2$	A	$(\theta_{23} : 101))$
	B	$(\theta_3 : 2), (\theta_7 : 1), (\theta_{22} : 34), (\theta_{25} : 64)$
$class_3$	A	$(\theta_0 : 37), (\theta_{10} : 65), (\theta_{12} : 160), (\theta_{17} : 1), (\theta_{28} : 126)$
	B	$(\theta_{16} : 389)$
$class_4$	A	$(\theta_{10} : 76)$
	B	$(\theta_{25} : 76)$
	C	$(\theta_{15} : 76)$
$class_5$	A	$(\theta_{10} : 248)$
	B	$(\theta_{22} : 61), (\theta_{25} : 187)$
	C	$(\theta_{21} : 248)$
$class_6$	A	$(\theta_4 : 214), (\theta_5 : 263), (\theta_8 : 79), (\theta_{11} : 175), (\theta_{13} : 193), (\theta_{14} : 2), (\theta_{29} : 90)$
	B	$(\theta_{18} : 593), (\theta_{19} : 90), (\theta_{29} : 333)$
	C	$(\theta_8 : 90), (\theta_{19} : 926)$
	D	$(\theta_{24} : 1016)$
	E	$(\theta_{20} : 1016)$
$\vdots$		

Table 3.2: Fragment from the list of node equivalence classes of the BLOCKCYL object

3.5 Pose estimation

The pose estimation problem is to determine the most probable pose –or the viewing coordinates– of a 3D object from an unknown 2D image of the object. The process can be split into two steps, first determining the aspect of the object and then refining the viewing parameters within the extents of the aspect until a satisfactory match is found. During the first step, when the whole viewing space needs to be considered, a faster and simpler matching is applied, while in the second step the range of possible viewing coordinates are limited to a few small areas defined by the proposed aspect(s), which allows for more expensive –and more accurate– matching algorithms.

The reference classes discussed in the previous chapter can be utilised to find candidates for the aspect of the object, using a simple and efficient method described below. Determining the precise viewing coordinates within the aspect has not been part of this research, but all image, high level feature and graph matching data is kept throughout the model building process. This data could be used for refining the viewing coordinates within the aspect.

3.5.1 Matching reference classes

The image from the unknown pose is submitted to the same preprocessing and feature extraction steps which is employed for building the model, and the image contours are assigned contour types using the k-means classifier from the clustering. An image consisting of n contours is thus converted to a list of n contour types:

$$unk = \{\theta_{i_1}, \theta_{i_2}, \ldots, \theta_{i_n}\} \tag{3.7}$$

where $0 \leq i_1, \ldots, i_n \leq k - 1$.

The unknown pose is then matched to all reference classes using simple or weighted bipartite matching [83]. The details of the matching are best explained through an example. Consider the class $class_1$ from (3.4), and unknown pose defined by $unk_1 = \{\theta_{20}, \theta_{23}, \theta_4, \theta_9, \theta_{26}\}$. The graph for matching the unknown pose to the reference class is shown on Fig. 3.10. The nodes $1 \ldots 5$ correspond to the contours of the unknown image, and the nodes $A \ldots E$ to the contours of the reference class. There is an edge between the nodes i and X, if the type of contour i is on the list for contour X in the reference class. The matching is either binary (just requiring the existence of the edge), or weighted using one of the schemes from Section 3.4.4, in which case the sum of the weights is to be maximised in addition to match as many nodes as possible. Figure 3.10b shows the bipartite graph with the weights from (3.6), where the sum of the weights for the best match is 2.3.

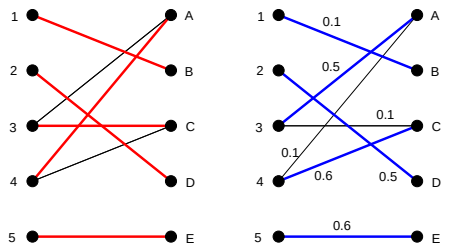


Figure 3.10: (a)Simple and (b)weighted bipartite matching

The selection of the best candidate classes is based on a combination of a number of factors:

- the number of nodes matched

- the number of unmatched nodes in the unknown image
- the number of unmatched nodes in the reference class
- the weight of the best match

A simple scoring function, which is not using the weight so it is applicable together with the simple bipartite matching is:

$$score = 10 \cdot m - |n_{unk} - n_{ref}| \tag{3.8}$$

This scheme, which prefers classes with the most matched contours and slightly penalises classes where the number of contours is different from the number of contours in the unknown, can be extended to include the weights:

$$score = \alpha \cdot m - \beta \cdot |n_{unk} - n_{ref}| + \sum_{\forall i,j:M_{ij}=1} w_{ij} \tag{3.9}$$

where m is the number of contours matched, n_{unk} and n_{ref} is the number of contours in the unknown image and the reference class, w_{ij} is the weight of the match between contour i of the unknown image and contour j of the reference class, M is the matrix corresponding to the best weighted bipartite match between the unknown image and the reference class, and α, β are appropriate constants. If α and β satisfy $\alpha \gg \beta \gg \sum w_{ij}$, the results will be similar to using (3.8), but in case of two equally good candidates the largest one is preferred.
(Note that the weights do not indicate the goodness of a match, but how large area (or number of views) of the viewing sphere a particular type of contour is visible within the class, and therefore relates to the probability of a view belonging to a class given that some contour type is observed.)

3.5.2 Multiple object classes

It is possible to extend the above procedure from a single object to multiple object classes. The pose (aspect) of the unknown object is estimated using each of the reference models assuming that it belongs to that object class. The scores for the best pose candidates are compared across the objects classes. If the object indeed belongs to one of the reference object classes, it is more likely that good pose estimates are found when the correct model is used, compared to using the other models. The scores give therefore a good indication of the object class to which the unknown object belongs to.

This method is simple and straightforward to implement based on the single class pose estimation. However, the model for each reference object class needs to be built in advance, which may be time consuming if there are a lot of object classes (though this is done only once). Also, a single class pose estimation is computed for each reference model. This is probably not a serious problem, as the pose estimation procedure is very fast once the model has been built, but if necessary one could use some sort of indexing or hashing technique across the different models to improve the performance.

Section 4.4.2 presents experimental results for multi-class pose estimation.

3.6 Interface to recognition system

The fast pose (aspect) estimation procedure makes the described model suitable for hypothesis generation in a (real-time) recognition system, while the recognition system does the verification of the correctness of the proposed candidates. The time consuming part of the process, the model building is performed off-line, and only needs to be done once. The amount of data produced by the model building may be very large, but for the aspect estimation uses only a compact representation with negligible size (typically 10-100kiB). In case refinement of the viewing coordinates within the aspect is implemented by the pose estimation, access to all matching and feature data may be required. Another option is to provide access to this data by the recognition system.

Chapter 4

Results

In the course of computing the results presented in this chapter, 270667 ray tracing and feature extraction operations, and 728156 graph matching operations have been performed.

4.1 The objects

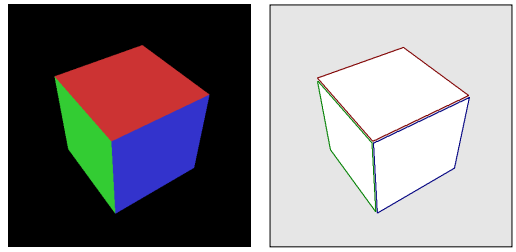
The experiments have been performed using a number of different objects, seven of them (CUBE, HOLEBLOCK, DUMBBELL, L_SHAPE, BLOCKCYL, PEGBLOCK, TRUCK) presented here.

The complexity of the objects varies from a simple cube (boundary is given by six squares) to the model of the TRUCK object with a boundary consisting of 65 polygons, 12 rings and 6 cylinders).

4.1.1 Virtual objects

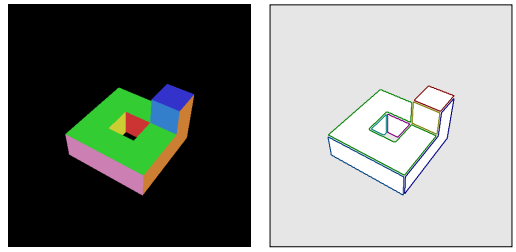
Some of the objects only exist as 3D models, and there is no corresponding real item (CUBE, Figure 4.1; HOLEBLOCK, Figure 4.2; DUMBBELL, Figure 4.3). These objects are included here in order to demonstrate the features of the modelling method. The boundary representation of these objects consists of:

CUBE	6 polygons (squares)
HOLEBLOCK	13 polygons
DUMBBELL	4 rings and 3 cylindrical surfaces



(a) (b)

Figure 4.1: CUBE: Simple cube object, ray traced(a) and segmented(b)

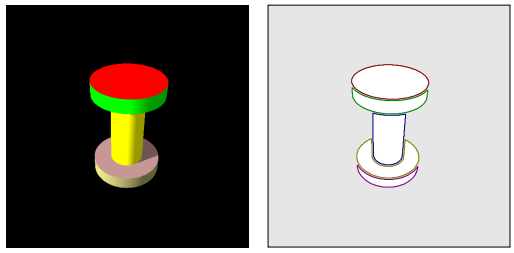


(a) (b)

Figure 4.2: HOLEBLOCK: Block object with a hole through it. Ray traced(a) and segmented(b) image.

4.1.2 Real objects

The following objects (L_SHAPE, BLOCKCYL, PEGBLOCK, TRUCK) are real test objects with 3D models based on actual measurements. The model of the TRUCK is somewhat simplified, with some angles replaced by right angles and some surface detail left out. The surface colours of all four objects have been obtained by averaging colour samples from images taken with an RGB camera. The BLOCKCYL object has been modelled both with original colours (BLOCKCYL1) and with unique colours for each face (BLOCKCYL2), the latter also with a different (shorter) camera-to-object distance (BLOCKCYL3) (though with a wider lens angle, so the object appears smaller on the ray traced images).

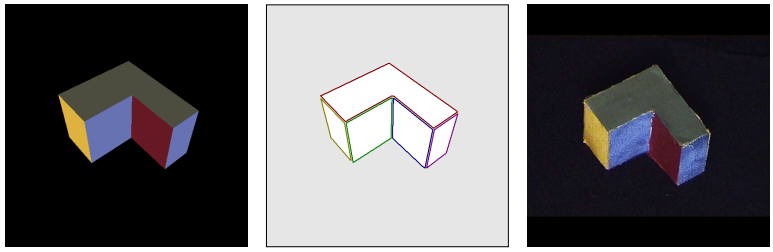


(a) (b)

Figure 4.3: DUMBBELL: Dumbbell object, constructed from three cylinders, ray traced(a) and segmented(b).

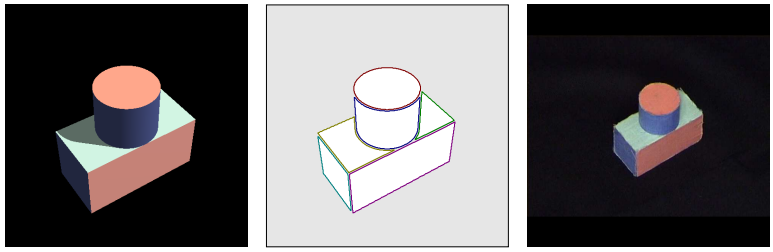
These objects have been used for pose estimation from both synthetic and real images. The boundary characteristics are the following:

L_SHAPE	8 polygons
BLOCKCYL	6 polygons, 1 ring and 1 cylindrical surface
PEGBLOCK	6 polygons, 1 ring and 1 cylindrical surface
TRUCK	65 polygons, 12 rings and 6 cylindrical surfaces



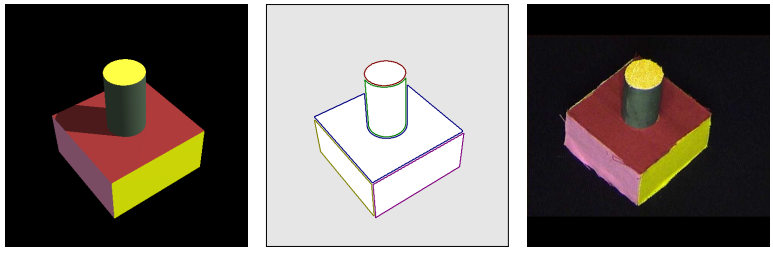
(a) (b) (c)

Figure 4.4: L_SHAPE: Simple L-shaped object. Ray traced(a) and segmented(b) images of the model and a colour image the real object(c).



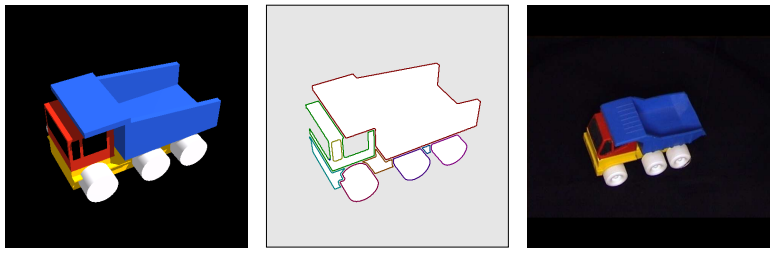
(a) (b) (c)

Figure 4.5: BLOCKCYL: A rectangular block with a cylinder mounted on top. Ray traced(a) and segmented(b) images of the model and a colour image the real object(c).



(a) (b) (c)

Figure 4.6: PEGBLOCK: A flat square block with a peg mounted on top. Ray traced(a) and segmented(b) images of the model and a colour image the real object(c).



(a) (b) (c)

Figure 4.7: TRUCK: Plastic childrens toy truck object by Spindler Toys, West Germany. Ray traced(a) and segmented(b) images of the model and a colour image the real object(c).

4.2 Model generation statistics

4.2.1 Model generation time

Table 4.1 summarises the model generation time for different objects. The column *feature* shows the number of ray tracing and feature extraction calls, and the column *graph* shows the number of graph matching calls (which is about 65% more than the actual graph matching operations executed, since the result of a potential reverse match ($\mathrm{match}(B, A)$ instead of $\mathrm{match}(A, B)$) is reused[1] if that has been computed earlier).

The hardware configuration of the computers used for building the models is indicated in the *hw* column:

(a) Intel Pentium M 1.2GHz CPU, 640MB memory, Linux 2.6.4
(b) Intel Pentium 4 2.4GHz CPU, 1GB memory, Linux 2.6.4
(c) AMD Athlon XP 2400+ CPU, 1GB memory, Linux 2.6.4
(d) Dual Intel Xeon 3.06GHz CPU, 4GB memory, Linux 2.4.26 (using only one of the CPUs for modelling)

The modelling system is CPU-bound, and it has only moderate memory and disk requirements (100-150 megabytes of memory and 4-700 megabytes of disk space for a typical level 6 model, although the data generated for the level 7 model of the truck consumes close to 5 gigabytes). The performance could significantly be increased by distributing the task to several processors – which is quite feasible as the different processors could work on separate areas of the viewing space independently. The current implementation supports placing the ray tracing, the feature extraction, the graph matching and the main control processes on four different hosts.

All image, feature and matching data generated for a level n model is preserved in databases, so it can be reused for building a level $n+1$ or higher model; also it is used for clustering and computing the equivalence classes. Some of the models were built this way, in which case the incremental time is indicated.

[1] Graph matching is only called if the two graphs have the same number of nodes, so the assumption that the match cost is symmetric is reasonable. Otherwise, missing a node from the reference graph may have a different cost than missing a node from the unknown graph.

object	level	feature	graph	hw	time
CUBE	2	162	402	(a)	00h 10m 42s
CUBE	3	642	2706	(a)	00h 26m 46s*
CUBE	4	2212	11094	(a)	01h 31m 29s*
CUBE	5	4941	21252	(a)	02h 43m 55s*
CUBE	6	9759	37119	(a)	04h 34m 10s*
CUBE	7	19587	69747	(a)	09h 32m 15s*
HOLEBLOCK	6	23708	103038	(b)	24h 15m 15s
DUMBBELL	6	17567	93243	(c)	13h 52m 51s
L_SHAPE	5	7624	38373	(a)	07h 25m 28s
L_SHAPE	6	15832	70227	(d)	10h 11m 27s
BLOCKCYL1	6	18510	91503	(c)	15h 39m 13s
BLOCKCYL2	6	18039	92463	(c)	15h 33m 28s
BLOCKCYL3	6	14863	66558	(c)	11h 56m 29s
PEGBLOCK	6	16400	76632	(c)	12h 38m 17s
TRUCK	6	38039	167964	(c)	137h 42m 56s
TRUCK	7	126161	569679	(c)	356h 21m 50s*

Table 4.1: `dyn` model generation statistics for different objects.
*Incremental, starting with data from previous level

4.3 Details and evaluation of the model

The highest level of subdivision was 6 for most objects. At this level over 90% of the viewing sphere is covered by accepted patches for all test objects except the HOLEBLOCK (80%) and the TRUCK, where the level 6 and 7 models cover 62% and 72% respectively. Considering the complexity of the objects and the number of viewing sphere partitions discovered, this coverage is very good.

Tables 4.2 to 4.4 and 4.6 to 4.12 show detailed statistics of the model building process for the objects, at several levels of icosahedron subdivision (2–6 for most objects, up to level 7 for some), using different threshold values for graph matching. The columns contain the following data (common for all tables):

object The name of the object.

level Level of icosahedron subdivision.

threshold Acceptance threshold for graph matching. In case two values are given (e.g. 10000/100000), the first one was used for patch subdivision, and the second for clustering the patches after the desired resolution has been reached.

triangles	Total number of triangles at completion. The number of triangles in the worst case is $t_l = 20 \cdot 4^l$, where l is the level of subdivision ($t_1 = 80, t_2 = 320, t_3 = 1280, t_4 = 5120, t_5 = 20480, t_6 = 81920, t_7 = 327680$).
feature	Number of ray tracing and feature extraction calls.
graph	Number of graph matching calls. The number of graph matching operations actually executed is shown in parentheses after the highest level results with the lowest threshold – this is less then the total number of graph matching calls, as the results for a reversed match is used in case that has been computed earlier.
part	Number of patch clusters.
area	The area covered by all patch clusters relative to the entire surface of the viewing sphere.

The result of the modelling at the highest level of subdivision is shown on Figures 4.9 to 4.20. Each figure (except the first) contains six projections of the viewing sphere (Fig. 4.8):

- top-left: the $z \geq 0$ hemisphere to the xy plane
- top-right: the $z < 0$ hemisphere to the xy plane
- middle-left: the $x < 0$ hemisphere to the yz plane
- middle-right: the $x \geq 0$ hemisphere to the yz plane
- bottom-left: the $y < 0$ hemisphere to the zx plane
- bottom-right: the $y \geq 0$ hemisphere to the zx plane

The small overlayed images show the object as seen from the middle point on the hemisphere (coordinates $(0, 0, 1)$, $(0, 0, -1)$, $(-1, 0, 0)$, $(1, 0, 0)$, $(0, -1, 0)$ and $(0, 1, 0)$)).

4.3.1 CUBE

The CUBE object has 26 different aspects (6 from each side, when only one face is visible, 12 looking at the edges with two faces visible, and 8 from the corners with three visible faces. Table 4.2 shows that the level 7 model, with graph matching threshold relaxed from 10_4 to 10_5 for the clustering phase gives exactly 26 clusters. Figure 4.9a shows how the viewing sphere is partitioned into clusters seen from the positive z coordinate axis. Due to the symmetry of the cube the other five views are very similar, though there is some variation along the boundaries, since the icosahedron based tessellation does not have the same symmetry as the cube. Figure 4.9b, and an enlarged detail on Figure 4.9c shows the same side of the viewing sphere,

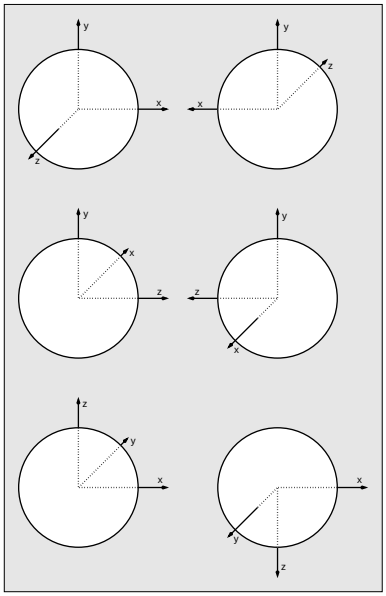


Figure 4.8: Six projections of the viewing sphere

with the accepted patches in red. The adaptive nature of the tessellation is clearly visible, the accepted patches are larger near the centre of the clusters, and get smaller as one moves toward the edges, pin-pointing the cluster boundaries with increasing precision.

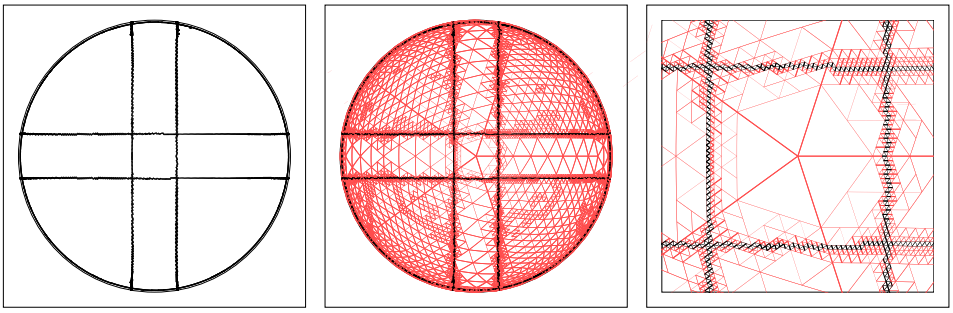
At lower levels, the number of clusters is first smaller than the expected 26 (levels 2 and 3), then it gets larger (levels 4-7). Even at level 7 there are 35 clusters, unless the graph matching threshold is increased. The reason for this is that at low levels the resolution is just not enough to give even a single accepted patch for each clusters. Large clusters show up relatively early, but smaller ones obviously need more subdivisions. As the resolution increases, more clusters are discovered. Some aspects of the object are represented by several clusters which are separated by patches where the match is not accepted yet. These clusters will merge later at higher resolutions (note the number of clusters decreasing from 44 at level 4 to 28 at level 5). At the same time, new clusters may be introduced along the boundaries (29 and 35 clusters in total at levels 6 and 7). These clusters are relatively small in size and are easily eliminated by keeping only clusters larger than a threshold (which may be a percentage of the largest cluster discovered), or by increasing the graph matching tolerance for the clustering phase.

The accumulated area of the clusters is increasing for each level, covering 97.2% of the surface

at level 7 with the matching threshold set to 10^5 for clustering. In the ideal case, when all unaccepted patches cross the boundary between two aspects, the number of these patches is roughly doubled, while the area of a single patch is about one fourth of the area of a patch at the previous level. This means that the uncovered area would be halved for each iteration. We can observe this tendency for the cube, where the uncovered area from level 4 to level 7 is 27.6%, 12.3%, 6.3% and 3.7%, respectively.

object	level	threshold	triangles	feature	graph	part	area
CUBE	2	10000	320	162	402	0	0
CUBE	3	10000	1280	642	2706	20	0.250781
CUBE	4	10000	4157	2212	11094	44	0.723828
CUBE	5	10000	8399	4941	21252	28	0.877051
CUBE	6	10000	15953	9759	37119	29	0.937390
CUBE	7	10000	31340	19587	69747	35	0.962787
					(45382)		
CUBE	7	10k/100k	31340	19587	69747	26	0.972113

Table 4.2: `dyn` model generation statistics for the CUBE object



(a) (b) (c)

Figure 4.9: Viewing sphere partitions for the cube object, level=7, threshold=10000,100000. (a)Partitions boundaries, (b)partition boundaries with accepted patches shown in red, (c)surface detail showing the adaptive subdivision

4.3.2 HOLEBLOCK

The HOLEBLOCK object (Table 4.3) is clearly more complex than the CUBE. The area covered by the clusters is only 81% at level 6 (vs. 93% for the CUBE at that level). The number aspects (and so the number of clusters) is much higher, so total length of the boundaries is longer, which means more patches crossing boundaries – which is the main reason for

covered area being smaller. Figure 4.10 shows that most clusters are quite clear, with narrow boundaries, except a few narrow areas where two boundaries run near each other.

Unlike in the case of the CUBE, it is not at all trivial to see which aspects of the object are observed from the different areas of the viewing sphere. Some visual events are easy to recognise, but overall picture becomes quite complex in general, mainly due to self-occlusions.

object	level	threshold	triangles	feature	graph	part	area
HOLEBLOCK	2	10000	320	162	252	0	0
HOLEBLOCK	3	10000	1280	642	1680	0	0
HOLEBLOCK	4	10000	5120	2562	10683	99	0.098632
HOLEBLOCK	5	10000	18965	9830	49656	182	0.617432
HOLEBLOCK	6	10000	42470	23708	103038 (61722)	301	0.775403
HOLEBLOCK	2	30000	320	162	252	0	0
HOLEBLOCK	3	30000	1280	642	1680	24	0.156250
HOLEBLOCK	4	30000	4520	2353	8310	98	0.542383
HOLEBLOCK	5	30000	11549	6511	20193	159	0.694482
HOLEBLOCK	6	30000	30320	17655	55539	286	0.800220
HOLEBLOCK	2	100000	317	162	240	10	0.215625
HOLEBLOCK	3	100000	1070	573	840	17	0.332813
HOLEBLOCK	4	100000	3632	1960	4758	100	0.557813
HOLEBLOCK	5	100000	10424	6001	15750	161	0.711865
HOLEBLOCK	6	100000	28127	16639	47184	280	0.818652
HOLEBLOCK	6	10k/100k	42470	23708	103038	295	0.812061

Table 4.3: `dyn` model generation statistics for the HOLEBLOCK object

4.3.3 DUMBBELL

The DUMBBELL object (Table 4.4) is a solid of revolution, so we expect the model to have the same rotational symmetry – which is indeed the case on Figure 4.12. The cluster boundaries for the DUMBBELL form 14 circles with centre on the y-axis, dividing the viewing sphere into 15 areas. This is 6 more than the theoretical 9 aspects (8 boundaries corresponding to the visual events on Figure 4.11). The reason is that due to the limited resolution of the camera (or the simulated camera, ray tracing) accidental views [140, 141] are visible over finite areas of the viewing sphere. For example, when moving from the top region (region 1) of the viewing sphere on Figure 4.11 to region 2, the surfaces B, E and F should become visible at the same time. In fact what happens is that these surfaces become visible one-by-one, first E, then B and F at last. The image contours corresponding to these surfaces have different sizes, so it will be at different angles when the size of image contour (number of pixels

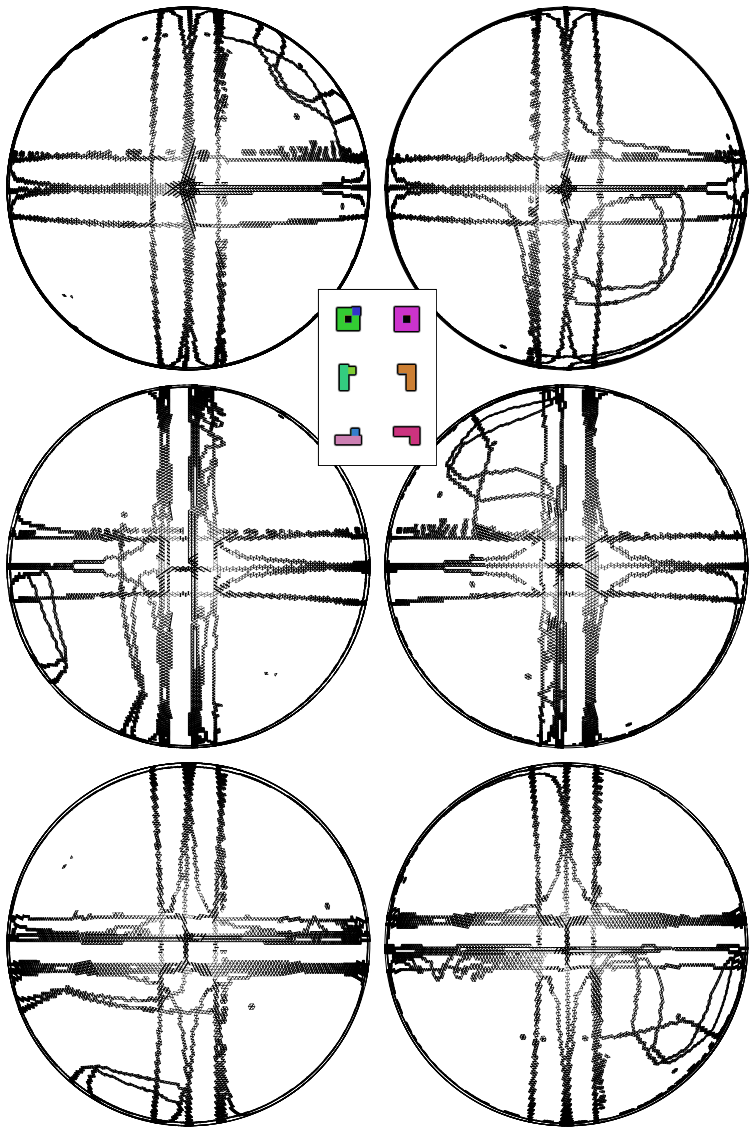


Figure 4.10: Viewing sphere partitions for the HOLEBLOCK object, level=6, threshold=10000,100000

enclosed) representing these surfaces will be large enough for the contour not to be discarded by feature extraction.

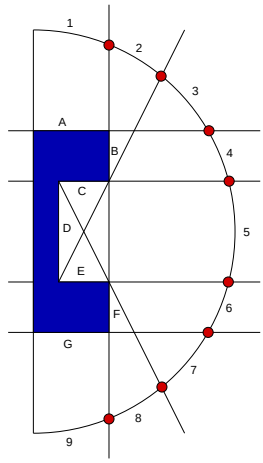


Figure 4.11: Expected visual events for the DUMBBEL object.

The clusters for the DUMBBELL at level 6 cover over 90% of the viewing surface. The number of clusters is 231, with many small clusters for the accidental views.

4.3.4 L_SHAPE

The complexity of the L_SHAPE object is somewhere between the CUBE and the HOLEBLOCK, and this is reflected in the modelling results (Figure 4.13 and Table 4.6). Some of the faces of the object have similar colours (so the corresponding contours in the image are not separable), and it influences the symmetry of the partitions. (The effect will be more obvious for the BLOCKCYL and PEGBLOCK objects.)

At level 6, the model covers nearly 90% of the viewing sphere, and the number of partitions is 125, out of which 47 are relatively large, covering from 6.17% of the sphere to 0.27%, and the rest (about 2/3) are rather small (all below 0.01%), many consisting of only one or two patches (Table 4.5). The reason for the large number of small partitions is some "noise" near the boundaries, and narrow partitions (possibly corresponding to accidental views) which haven't been merged together yet.

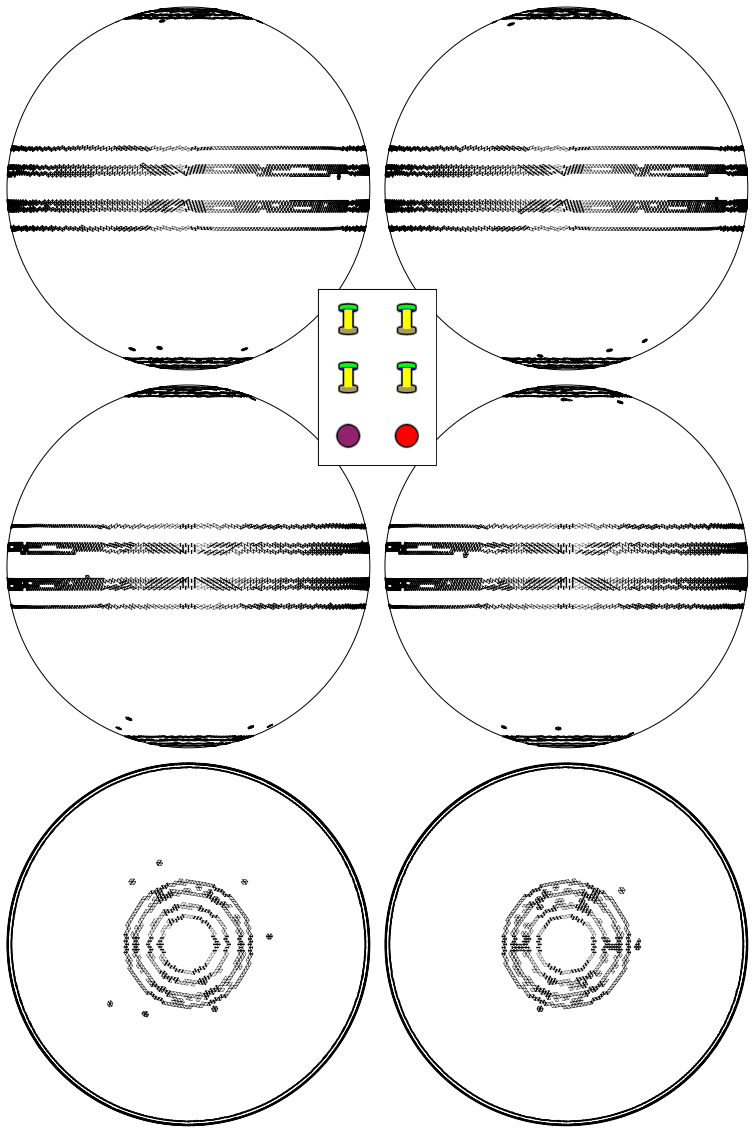


Figure 4.12: Viewing sphere partitions for the dumbbell object, level=6, threshold=10000,100000

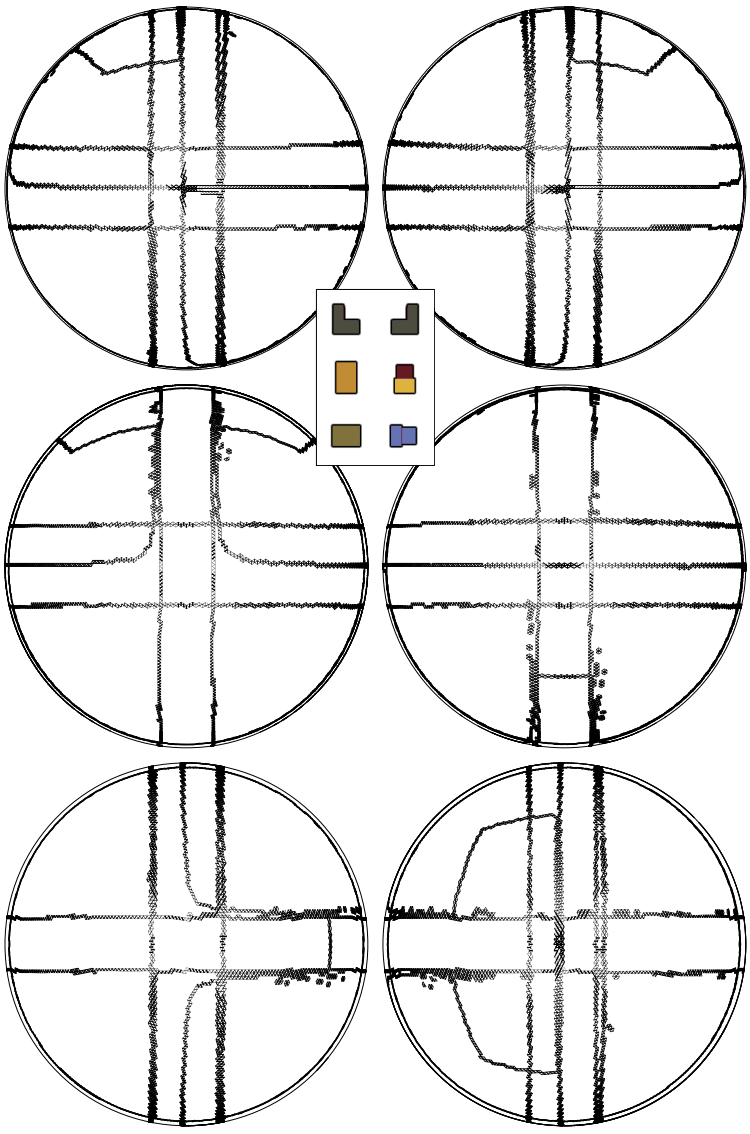


Figure 4.13: Viewing sphere partitions for the L_SHAPE object, level=6, threshold=10000,30000

object	level	threshold	triangles	feature	graph	part	area
DUMBBELL	2	10000	320	162	780	0	0
DUMBBELL	3	10000	1280	642	3312	0	0
DUMBBELL	4	10000	5120	2562	15174	137	0.205859
DUMBBELL	5	10000	17318	9221	55395	61	0.763525
DUMBBELL	6	10000	31847	17567	93243 (52842)	252	0.869910
DUMBBELL	2	30000	320	162	780	0	0
DUMBBELL	3	30000	1280	642	3312	2	0.562500
DUMBBELL	4	30000	2960	1550	6534	88	0.751563
DUMBBELL	5	30000	6776	3863	13236	47	0.838281
DUMBBELL	6	30000	16712	9787	32994	329	0.891272
DUMBBELL	2	100000	224	130	396	2	0.512500
DUMBBELL	3	100000	692	398	960	2	0.634375
DUMBBELL	4	100000	2096	1162	3078	72	0.765625
DUMBBELL	5	100000	5696	3366	8925	46	0.851172
DUMBBELL	6	100000	14840	8953	25692	228	0.911316
DUMBBELL	6	10k/100k	31847	17567	93243	231	0.909338

Table 4.4: `dyn` model generation statistics for the DUMBBELL object

4.3.5 BLOCKCYL

The phenomenon of distinct contours not being separated during image segmentation due to similar colour ratios, which has been mentioned for the L_SHAPE object, becomes even more obvious if we look at the BLOCKCYL object. The original object (BLOCKCYL1) has the same colour on the cylindrical surface as on the two small ends of the block, corresponding to the real object. A second model of the object has been coloured using unique colours for each surface. The difference between the viewing sphere partitioning results is easy to see on Figures 4.15 and 4.16. Figure 4.14 shows one of the views with results from both objects, common boundaries in black, BLOCKCYL1 boundaries in red and BLOCKCYL2 boundaries in blue.

The third set of results (BLOCKCYL3, Figure 4.17) comes from the same model as the second, but with the camera-to-object distance decreased[2]. The result of the different perspective is some shifting of the partition boundaries, while the overall layout is still very similar. The camera-to-object distance plays an important factor for the aspects of the object when the camera is placed near the object. The closer the camera is to the object, the stronger the effect of the perspective projection. At longer distances the perspective effect decreases as the

[2]Actually the camera is still placed on the surface of the Gaussian sphere, but the object has been scaled up so it appears closer.

id	△	area (%)	id	△	area (%)	id	△	area (%)	id	△	area (%)	id	△	area (%)
22	879	6.174316	13	145	1.462402	86	23	0.028076	78	2	0.002441	111	2	0.002441
17	845	6.074219	2	275	1.459961	61	13	0.015869	77	2	0.002441	110	2	0.002441
42	1800	5.914307	43	281	1.437988	70	12	0.014648	75	2	0.002441	107	2	0.002441
32	1542	5.756836	5	127	0.894775	84	11	0.013428	66	2	0.002441	106	2	0.002441
37	1654	5.651855	11	128	0.881348	116	10	0.012207	65	2	0.002441	105	2	0.002441
40	1617	5.599365	14	89	0.826416	89	8	0.009766	64	2	0.002441	104	2	0.002441
26	1094	4.213867	39	222	0.670166	79	8	0.009766	63	2	0.002441	103	2	0.002441
35	1050	3.958740	45	209	0.665283	52	8	0.009766	62	2	0.002441	102	2	0.002441
3	206	2.441406	27	187	0.660400	69	6	0.007324	60	2	0.002441	101	2	0.002441
8	240	2.420654	25	184	0.598145	51	6	0.007324	58	2	0.002441	100	2	0.002441
4	222	2.373047	10	95	0.474854	72	5	0.006104	57	2	0.002441	99	1	0.001221
6	284	2.229004	9	72	0.417480	71	5	0.006104	56	2	0.002441	98	1	0.001221
29	463	2.084961	7	68	0.379639	68	4	0.004883	55	2	0.002441	97	1	0.001221
12	244	2.044678	47	77	0.306396	67	4	0.004883	54	2	0.002441	93	1	0.001221
1	317	2.001953	23	85	0.294189	59	4	0.004883	125	2	0.002441	92	1	0.001221
41	433	1.949463	16	51	0.289307	50	4	0.004883	124	2	0.002441	91	1	0.001221
18	288	1.831055	30	80	0.288086	120	4	0.004883	123	2	0.002441	88	1	0.001221
33	364	1.806641	46	85	0.286865	117	4	0.004883	122	2	0.002441	83	1	0.001221
24	369	1.735840	36	87	0.285645	96	2	0.002441	121	2	0.002441	81	1	0.001221
34	323	1.730957	19	67	0.279541	95	2	0.002441	119	2	0.002441	76	1	0.001221
21	334	1.678467	20	89	0.269775	94	2	0.002441	118	2	0.002441	74	1	0.001221
38	344	1.639404	44	95	0.266113	90	2	0.002441	115	2	0.002441	73	1	0.001221
15	305	1.540527	48	56	0.097656	85	2	0.002441	114	2	0.002441	53	1	0.001221
28	228	1.516113	87	68	0.083008	82	2	0.002441	113	2	0.002441	109	1	0.001221
31	302	1.507568	49	31	0.041504	80	2	0.002441	112	2	0.002441	108	1	0.001221

Table 4.5: Partition sizes after 6 iterations of the L_SHAPE object

projection approaches the orthographic projection, and the visible aspects are less sensitive to distance variation.

The model statistics are summarised on Tables 4.7 to 4.9. The number of partitions for the BLOCKCYL1 object is about 50% larger then for the other two. The reason for this is that segmentation errors are more likely to occur on the images of this object, since the colour separation is not so good as for the other objects.

4.3.6 PEGBLOCK

Despite the geometrical symmetry of the PEGBLOCK object (Table 4.10), the resulting clusters on Figure 4.18 are not completely symmetrical. This is caused by the same effect as we have observed by the BLOCKCYL object: two surfaces of the object (the cylindrical surface and one of the sides of the block) having similar colour, so these will not be separated during image segmentation. The result of the two image segments merged together is some cluster boundaries disappearing (as the clusters are merged), but also some new boundaries being introduced (best seen on the top-right view, from $(0, 0, -1)$).

Apart from this the images of the PEGBLOCK object segment very well, the number of partitions is 67 at level 6 using 10^4 and 10^5 for the subdivision and clustering threshold. There are only a few tiny clusters indicating noise or inappropriate resolution near the boundaries,

object	level	threshold	triangles	feature	graph	part	area
L_SHAPE	2	10000	320	162	249	0	0
L_SHAPE	3	10000	1280	642	2112	28	0.097656
L_SHAPE	4	10000	4745	2451	11547	72	0.403516
L_SHAPE	5	10000	13907	7624	38373	62	0.781885
L_SHAPE	6	10000	27308	15832	70227 (42376)	160	0.869727
L_SHAPE	2	30000	320	162	249	3	0.009375
L_SHAPE	3	30000	1271	642	2076	55	0.450781
L_SHAPE	4	30000	3380	1924	6096	50	0.691211
L_SHAPE	5	30000	8123	4930	15264	55	0.812012
L_SHAPE	6	30000	19673	12100	39969	136	0.887756
L_SHAPE	2	100000	311	162	213	11	0.234375
L_SHAPE	3	100000	1046	566	1176	55	0.485156
L_SHAPE	4	100000	3023	1771	4668	49	0.700586
L_SHAPE	5	100000	7622	4705	13287	53	0.824561
L_SHAPE	6	100000	18401	11493	35040	125	0.899426
L_SHAPE	6	10k/100k	27308	15832	70227	125	0.897827

Table 4.6: `dyn` model generation statistics for the L_SHAPE object

the majority (56) are between 0.1% and 6.44% in area, containing at least 25 patches. (The size of the remaining 11 partitions is below 0.05%).

4.3.7 TRUCK

As expected, the TRUCK shows much greater variation in the observations as the observer moves along the viewing sphere (Fig. 4.20). The number of partitions is a magnitude larger then for the other objects, 1051 at level 6 using $(10^4, 3 \cdot 10^5)$ threshold values, and 2706 at level 7 with the same thresholds (Table 4.12). At level 6, the 1051 partitions cover 62.04% of the viewing sphere, this increases to 72.28% at level 7. The size of the partitions decreases very rapidly (Table 4.11 and Fig. 4.19), so the search space could be reduced significantly by discarding the small partitions, without considerable loss in coverage. The vast majority of over 1500 new partitions introduced at level 7 are tiny consisting of just one or a few patches. This is confirmed by Table 4.11. At level 6, 31% of the partitions (328) is needed to provide 95% of the total coverage (which corresponds to 58.949% of the viewing sphere). At level 6, only 17% of the partitons (458) gives 95% of the coverage (68.669% of the sphere).

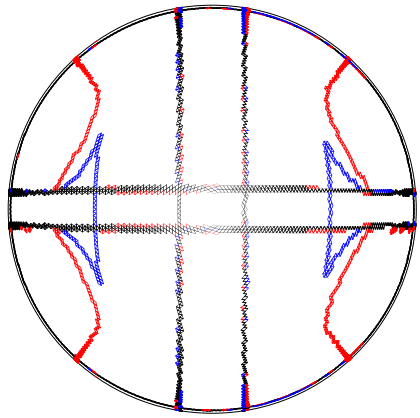


Figure 4.14: Differences between the viewing sphere partitions of the BLOCKCYL1 and BLOCKCYL2 objects due to unseparated image contours. Common boundaries in black, BLOCKCYL1 only in red and BLOCKCYL2 only in blue

object	level	threshold	triangles	feature	graph	part	area
BLOCKCYL1	2	10000	320	162	312	0	0
BLOCKCYL1	3	10000	1280	642	2343	17	0.050000
BLOCKCYL1	4	10000	4928	2515	12780	49	0.305664
BLOCKCYL1	5	10000	15593	8258	45777	136	0.725244
BLOCKCYL1	6	10000	32474	18510	91503 (53404)	157	0.869739
BLOCKCYL1	2	30000	320	162	312	6	0.031250
BLOCKCYL1	3	30000	1250	638	2223	45	0.313281
BLOCKCYL1	4	30000	3887	2148	8616	43	0.695703
BLOCKCYL1	5	30000	8561	5019	17649	66	0.813672
BLOCKCYL1	6	30000	20009	12061	41769	115	0.889868
BLOCKCYL1	2	100000	305	160	312	17	0.337500
BLOCKCYL1	3	100000	941	535	1185	33	0.550000
BLOCKCYL1	4	100000	2669	1601	4239	39	0.725586
BLOCKCYL1	5	100000	6884	4197	12081	58	0.835645
BLOCKCYL1	6	100000	16982	10481	32166	116	0.904675
BLOCKCYL1	6	10k/100k	32474	18510	91503	120	0.898499

Table 4.7: `dyn` model generation statistics for the BLOCKCYL1 object

object	level	threshold	triangles	feature	graph	part	area
BLOCKCYL2	2	10000	320	162	372	0	0
BLOCKCYL2	3	10000	1280	642	2544	17	0.035938
BLOCKCYL2	4	10000	4982	2533	13506	45	0.245117
BLOCKCYL2	5	10000	16577	8732	50814	105	0.751953
BLOCKCYL2	6	10000	31817	18039	92463 (53239)	109	0.891431
BLOCKCYL2	2	30000	320	162	372	2	0.006250
BLOCKCYL2	3	30000	1274	642	2520	35	0.340625
BLOCKCYL2	4	30000	3806	2084	8802	52	0.736914
BLOCKCYL2	5	30000	7847	4671	15993	54	0.842236
BLOCKCYL2	6	30000	17540	10766	35691	85	0.912415
BLOCKCYL2	2	100000	308	162	324	14	0.350000
BLOCKCYL2	3	100000	932	522	1152	26	0.550000
BLOCKCYL2	4	100000	2660	1573	4272	50	0.751563
BLOCKCYL2	5	100000	6476	4035	10665	56	0.852393
BLOCKCYL2	6	100000	15545	9814	28065	86	0.920166
BLOCKCYL2	6	10k/100k	31817	18039	92463	88	0.918323

Table 4.8: `dyn` model generation statistics for the BLOCKCYL2 object

object	level	threshold	triangles	feature	graph	part	area
BLOCKCYL3	2	10000	320	162	324	0	0
BLOCKCYL3	3	10000	1280	642	2346	25	0.090625
BLOCKCYL3	4	10000	4772	2463	12171	95	0.400977
BLOCKCYL3	5	10000	13973	7631	39504	67	0.807715
BLOCKCYL3	6	10000	25787	14863	66558 (39902)	92	0.892151
BLOCKCYL3	2	30000	320	162	324	8	0.050000
BLOCKCYL3	3	30000	1232	634	2154	33	0.496875
BLOCKCYL3	4	30000	3164	1795	5766	48	0.725195
BLOCKCYL3	5	30000	7385	4464	13344	60	0.838965
BLOCKCYL3	6	30000	17279	10665	33054	80	0.907959
BLOCKCYL3	2	100000	314	162	300	16	0.306250
BLOCKCYL3	3	100000	980	546	1146	33	0.517969
BLOCKCYL3	4	100000	2831	1660	4434	50	0.730273
BLOCKCYL3	5	100000	6974	4292	11700	64	0.845947
BLOCKCYL3	6	100000	16439	10291	29721	78	0.916394
BLOCKCYL3	6	10k/100k	25787	14863	66558	78	0.914917

Table 4.9: `dyn` model generation statistics for the BLOCKCYL3 object

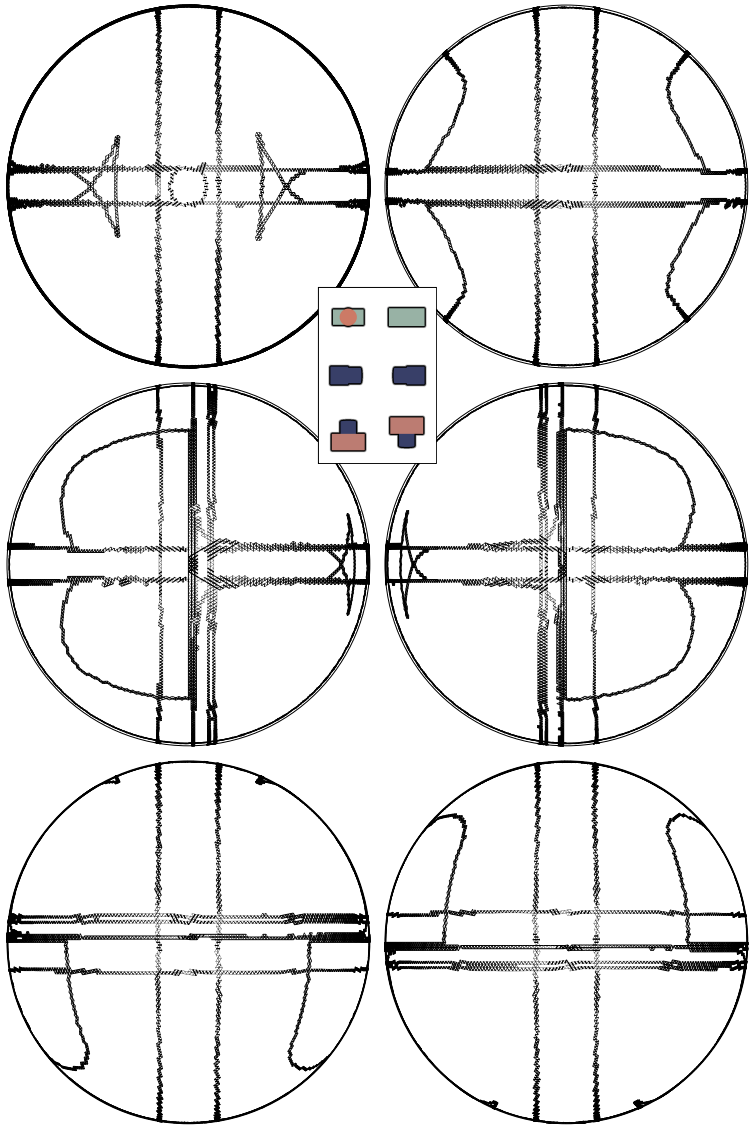


Figure 4.15: Viewing sphere partitions for the BLOCKCYL1 object, level=6, threshold=10000,100000

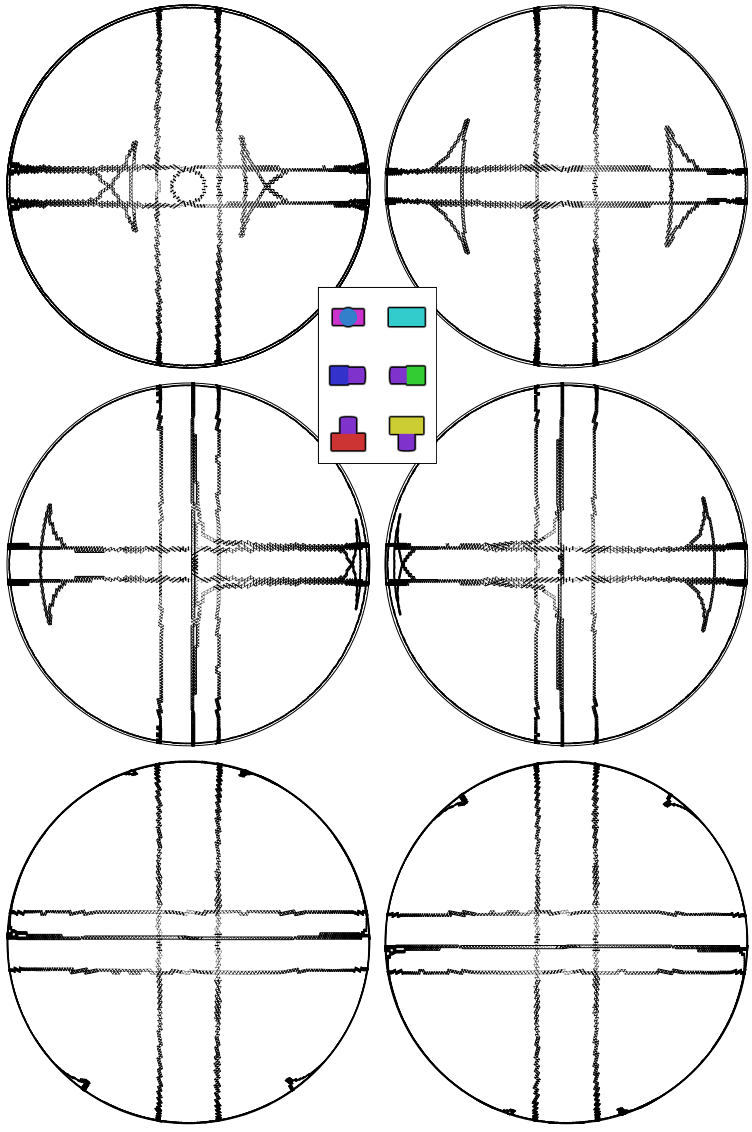


Figure 4.16: Viewing sphere partitions for the BLOCKCYL2 object, level=6, threshold=10000,100000

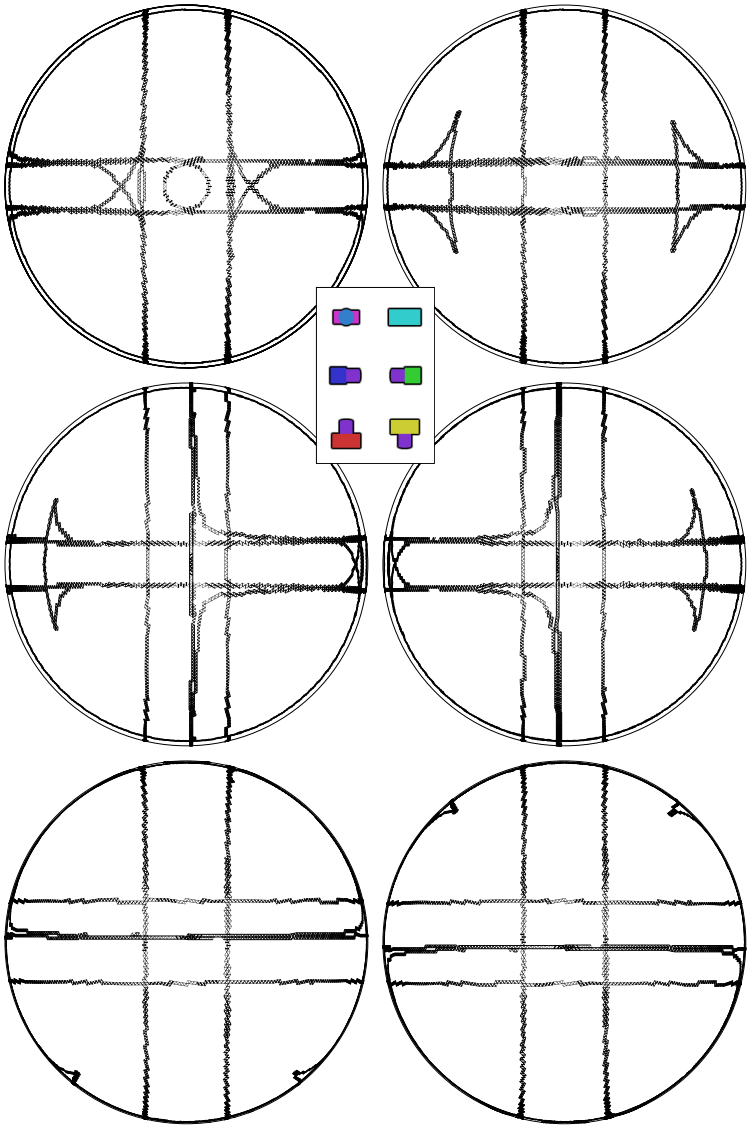


Figure 4.17: Viewing sphere partitions for the BLOCKCYL3 object, level=6, threshold=10000,100000

object	level	threshold	triangles	feature	graph	part	area
PEGBLOCK	2	10000	320	162	351	0	0
PEGBLOCK	3	10000	1280	642	2268	16	0.032812
PEGBLOCK	4	10000	4994	2535	12837	95	0.308789
PEGBLOCK	5	10000	15611	8427	45549	73	0.789160
PEGBLOCK	6	10000	28565	16400	76632 (45629)	94	0.877393
PEGBLOCK	2	30000	320	162	351	2	0.006250
PEGBLOCK	3	30000	1274	642	2244	41	0.454688
PEGBLOCK	4	30000	3368	1888	6333	55	0.689844
PEGBLOCK	5	30000	8132	4900	15633	64	0.812988
PEGBLOCK	6	30000	19622	12033	41019	69	0.891479
PEGBLOCK	2	100000	308	162	303	10	0.318750
PEGBLOCK	3	100000	962	532	1056	42	0.492188
PEGBLOCK	4	100000	2912	1704	4773	54	0.713477
PEGBLOCK	5	100000	7313	4522	13098	62	0.828906
PEGBLOCK	6	100000	17825	11122	35583	61	0.903833
PEGBLOCK	6	10k/100k	28565	16400	76632	67	0.899072

Table 4.10: `dyn` model generation statistics for the PEGBLOCK object

TRUCK, level 6			
a	b	c	d
10	105	47.736	78.703
20	210	55.452	89.379
30	315	58.693	94.603
40	420	60.165	96.976
50	525	60.874	98.119
60	630	61.300	98.806
70	735	61.556	99.219
80	840	61.783	99.585
90	945	61.912	99.791
100	1051	62.041	100.000

TRUCK, level 7			
a	b	c	d
10	271	64.267	88.915
20	542	69.568	96.250
30	812	70.915	98.113
40	1083	71.414	98.805
50	1353	71.697	99.195
60	1624	71.863	99.425
70	1895	72.028	99.654
80	2165	72.113	99.771
90	2436	72.196	99.886
100	2706	72.278	100.000

Table 4.11: Percentage of the surface area covered by the partitions, sorted partition size for level 6 and 7 models of the TRUCK. (a) percentage of partitions, (b) number of partitions, (c) percentage of viewing sphere, (d) percentage of covered area

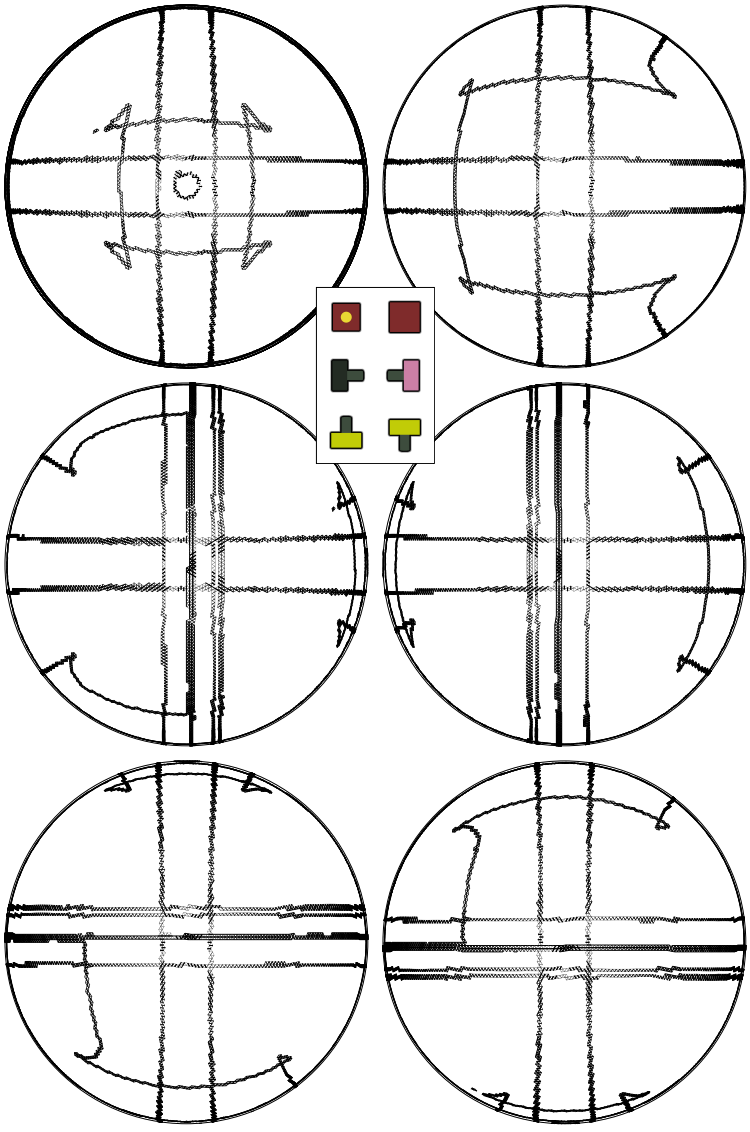


Figure 4.18: Viewing sphere partitions for the PEGBLOCK object, level=6, threshold=10000,30000

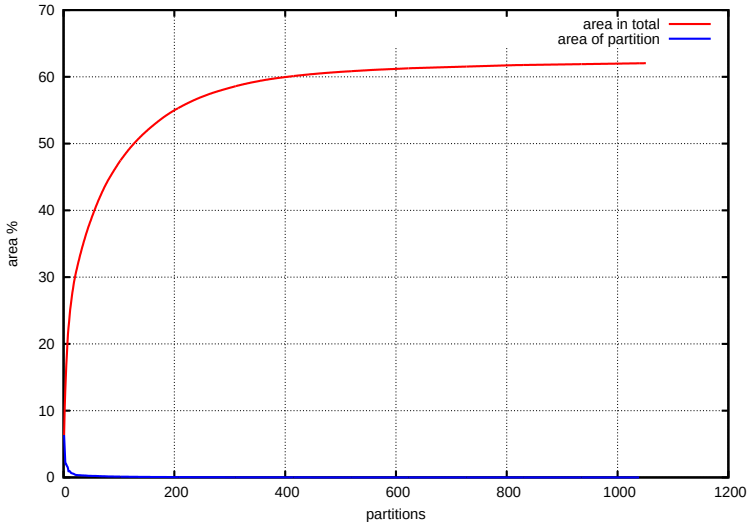


Figure 4.19: Percentage of the surface area covered by the partitions, sorted by partition size (TRUCK, level 6)

object	level	threshold	triangles	feature	graph	part	area
TRUCK	2	10000	320	162	132	0	0
TRUCK	3	10000	1280	642	1011	1	0.001563
TRUCK	4	10000	5114	2561	6393	25	0.024609
TRUCK	5	10000	20096	10130	36975	173	0.114941
TRUCK	6	10000	74474	38039	167964 (97661)	1372	0.338013
TRUCK	7	10000	237164	126161	569679 (333660)	4104	0.541272
TRUCK	2	30000	320	162	132	1	0.003125
TRUCK	3	30000	1277	642	999	4	0.051563
TRUCK	4	30000	4919	2480	5658	95	0.155859
TRUCK	5	30000	17885	9266	28572	408	0.366602
TRUCK	6	30000	56801	30805	102687	1117	0.538196
TRUCK	7	30000	170294	95355	331413	2749	0.665814
TRUCK	2	100000	317	162	120	5	0.046875
TRUCK	3	100000	1232	630	819	29	0.146094
TRUCK	4	100000	4511	2355	4260	149	0.294141
TRUCK	5	100000	15353	8343	20160	418	0.485010
TRUCK	6	100000	46994	26497	71127	964	0.640649
TRUCK	7	100000	135308	78278	221061	2253	0.747455
TRUCK	7	10k/100k	237164	126161	569679	2809	0.691595
TRUCK	6	10k/300k	74474	38039	167964	1051	0.620410
TRUCK	7	10k/300k	237164	126161	569679	2706	0.722787

Table 4.12: `dyn` model generation statistics for the TRUCK object

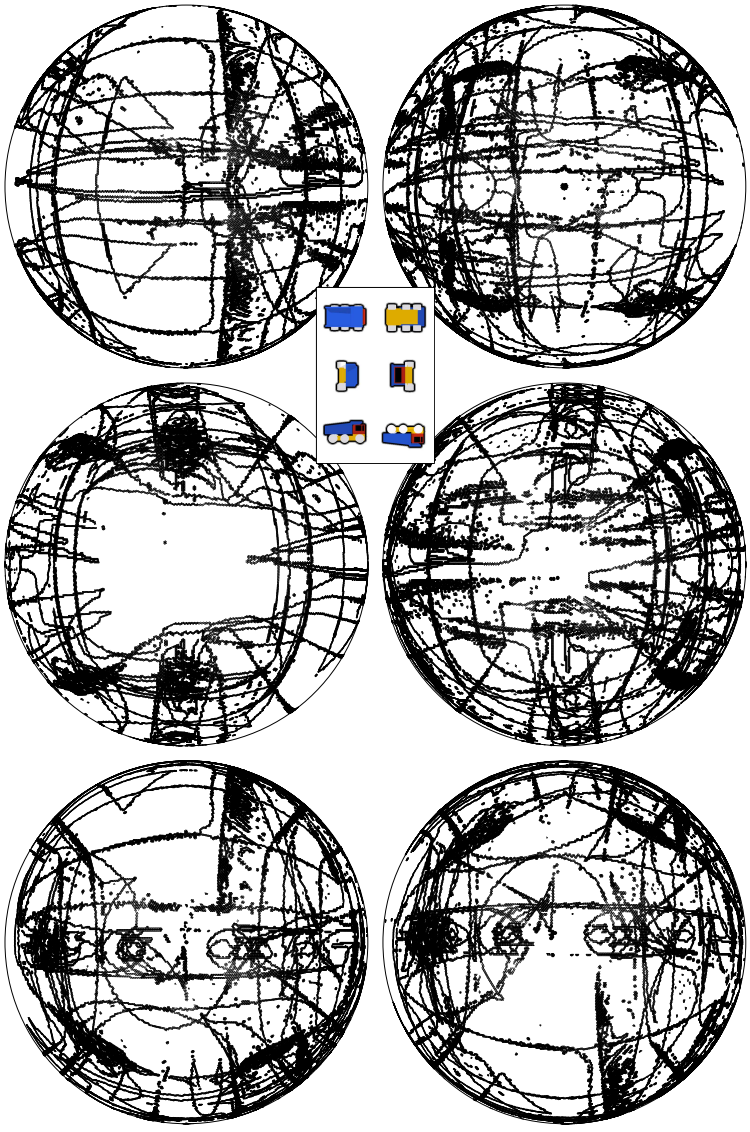


Figure 4.20: Viewing sphere partitions for the TRUCK object, level=7, threshold=10000,300000

4.4 Pose estimation from simulated images

For the first sequence of experiments, synthetic images were used as input. A set of n random viewpoints were selected, where $n = 100$ unless stated otherwise. The location of the viewpoints was uniformly distributed over the unit sphere[3] (Figure 4.21.). From each of these viewpoints ray traced images of the object were generated and the high level features were extracted.

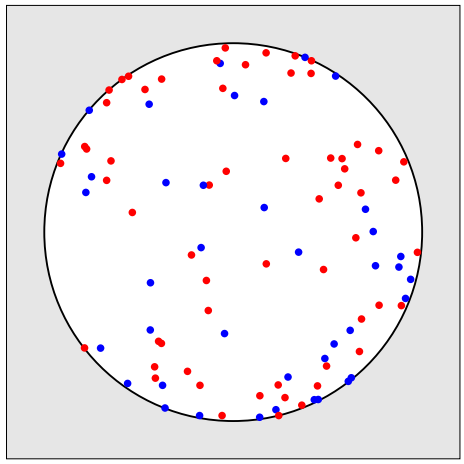


Figure 4.21: Distribution of the test views on the viewing sphere (for the TRUCK object). The red points are on the front ($z > 0$), and the blue points are on the back ($z \leq 0$).

4.4.1 Single class pose estimation

Using the high level features, the attribute vectors were computed for all contours of the unknown images, and the contours were assigned a contour type using the same k-means classifier which was also used for building the equivalence classes for the model (Section 3.4.3). The

[3] Using a method derived by Marsaglia [100], which consist of picking u and v from independent uniform distributions on $(-1, 1)$ and rejecting points for which $u^2 + v^2 \geq 1$. From the remaining points,

$$\begin{aligned} x &= 2u\sqrt{1-u^2-v^2} \\ y &= 2v\sqrt{1-u^2-v^2} \\ z &= 1-2(u^2+v^2) \end{aligned} \tag{4.1}$$

have a uniform distribution on the surface of a unit sphere.

pose of the object was estimated for each of the unknown images, and a shortlist of ten suggestions was built for the possible poses, using bipartite matching from the unknown contours (contour types) to each reference class, with a scoring function based on the second scoring scheme (number of views) described in Section 3.4.4 on page 34.

$$score = 10^6 \cdot m - 10^5 \cdot f(n_{unk}, n_{ref}) + \sum_{\forall i,j:M_{ij}=1} w_{ij} \tag{4.2}$$

where

$$f(x,y) = \begin{cases} y - x & \text{if} \quad y > x \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

This scoring function is very similar to (3.8), with the constants $\alpha = 10^6$ and $\beta = 10^5$ selected such that the most important contribution to the score comes from the number of contours matched, then from the penalty of the unknown image and reference class having different number of contours (only applied here if the reference class has more contours), and finally from the weights of the reference class ($\alpha \gg \beta \gg \sum w_{ij}$).

For the verification of the results, the correct pose was determined by computing the distance between the viewing coordinates of the unknown pose and each of the reference classes (defined as the minimum distance between the unknown and the views belonging to the class). The reference class with the smallest distance to the unknown was considered correct, although if the unknown viewpoint lies on the boundary between two classes, the nearest one may not necessarily be the correct one. (On the other hand, correct pose estimation results may be considered as a failures due to the same reason.) The results are evaluated based on the rank of the correct pose on the shortlist.

BLOCKCYL

At first sight the pose estimation results do not look very convincing, with only 22% correct first hit (Table 4.13). The reason for this is the symmetry of the object, which in addition to the geometric symmetry has been coloured such that the opposite faces have the same colour. The result is that several aspects will be indistinguishable from each other based on the available observations (Fig. 4.22). In addition, the pose estimation method is using only the contour parameters from the unknown images, and not the relations between the contours. This makes it insensitive to rotations, but also to mirroring. A closer study of the results where the

correct pose is at the 2nd to 5th position reveals that in all these cases the first suggested pose is from one of the "mirrored" aspects.

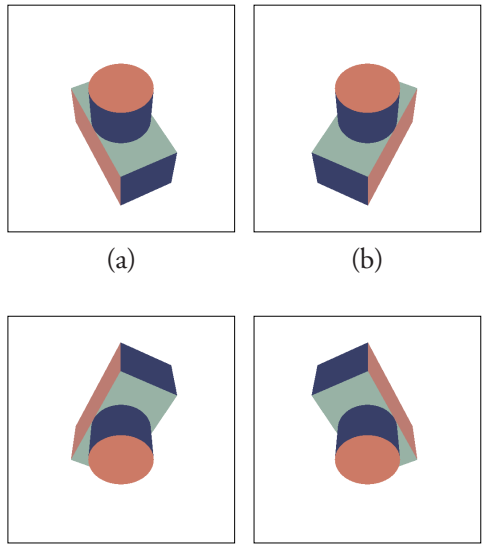


Figure 4.22: Indistinguishable aspects of the BLOCKCYL object. The viewing coordinates are a:(x, y, z) b:$(-x, y, z)$ c:$(x, -y, z)$ d:$(-x, -y, z)$, where $x = 0.541, y = 0.266, z = 0.798$

In six of the hundred cases the correct class was not amongst the first ten suggestions, these are listed in Table 4.14, and the corresponding images from the unknown pose together with a sample image from the proposed aspects are shown on Fig. 4.23. Common for all six cases that the unknown pose is very close to a boundary between two aspects, where one or more image contour is just about to appear or disappear. The potential for errors due to finite image resolution is also highest in these areas. Yet, even in these cases we find a relatively good match from one of the mirrored aspects.

Disambiguating the aspects by assigning unique colours to the different faces of the object (BLOCKCYL2) significantly improves the precision (Table 4.15).

Model parameters:

- level: 6

- subdivision threshold: 10000
- clustering threshold: 100000
- number of clusters: 120 (BLOCKCYL1) / 88 (BLOCKCYL2)
- total area covered by the clusters: 89.85% / 91.83%

position of correct suggestion	percentage of tests	accumulated percentage
1	22%	22%
2	21%	43%
3	18%	61%
4-5	26%	87%
6-10	7%	94%

Table 4.13: Single class pose estimation results for the BLOCKCYL1 object from synthetic images

viewing coordinates	correct class	dist	suggestions
$(+0.914, +0.405, +0.023)$	51	0.0301	14, 4, 6, 13, 115
$(-0.402, +0.226, +0.887)$	57	0.0067	30, 43, 37, 31, 36
$(-0.561, +0.054, +0.826)$	35	0.0153	17, 38, 43, 30, 31
$(-0.817, +0.566, +0.109)$	65	0.0093	55, 46, 45, 51, 29
$(-0.856, -0.516, +0.018)$	45	0.0262	14, 4, 6, 13, 115
$(-0.957, +0.113, +0.267)$	72	0.0064	36, 42, 37, 30, 31

Table 4.14: Incorrect suggestions for poses of the BLOCKCYL1 object. (dist is the distance between the unknown viewpoint and the nearest viewpoint within the correct class of the model)

position of correct suggestion	percentage of tests	accumulated percentage
1	98%	98%
2	1%	99%
6	1%	100%

Table 4.15: Single class pose estimation results for the BLOCKCYL2 object from synthetic images

L_SHAPE and PEGBLOCK

The L_SHAPE and PEGBLOCK objects are also symmetric, but due the combination of the shape and more varying colours the number of indistinguishable aspects is smaller. This

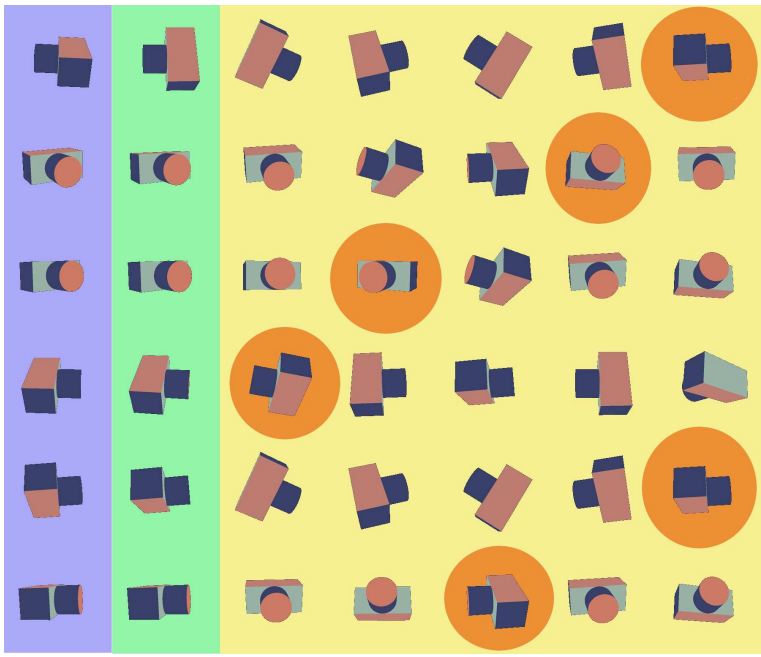


Figure 4.23: Suggested poses for some test views of the BLOCKCYL1 object. The unknown poses are on blue background, a random view from the correct class on green, and random views from the first five suggested classes on yellow. The darker circles indicate the suggestion which appears most similar.

is reflected in the results (Tables 4.16 and 4.17), with a precision somewhere between the BLOCKCYL1 and BLOCKCYL2 results.

Model parameters (L_SHAPE):

- level: 6
- subdivision threshold: 10000
- clustering threshold: 100000
- number of clusters: 125
- total area covered by the clusters: 89.78%

position of correct suggestion	percentage of tests	accumulated percentage
1	51%	51%
2	41%	92%
3	1%	93%
4	3%	96%
5	2%	98%

Table 4.16: Single class pose estimation results for the L_SHAPE object from synthetic images

Model parameters (PEGBLOCK):

- level: 6
- subdivision threshold: 10000
- clustering threshold: 100000
- number of clusters: 67
- total area covered by the clusters: 89.91%

position of correct suggestion	percentage of tests	accumulated percentage
1	45%	45%
2	47%	92%
3	3%	95%
4	2%	97%
5-9	3%	100%

Table 4.17: Single class pose estimation results for the PEGBLOCK object from synthetic images

TRUCK

For the TRUCK experiments, the level 6 model of the object was used with 1051 clusters in total. No clusters have been discarded even though over 60% consists of less than ten patches (whereas the largest clusters consist of more than a thousand). Even the large number of possible clusters does not decrease the precision significantly, though it is slightly worse than for the simple objects.

Model parameters:

- level: 6
- subdivision threshold: 10000
- clustering threshold: 300000
- number of clusters: 1051
- total area covered by the clusters: 62.04%

position of correct suggestion	percentage of tests	accumulated percentage
1	49%	49%
2	22%	71%
3	9%	80%
4-5	8%	88%
6-10	7%	95%

Table 4.18: Single class pose estimation results for the TRUCK object from synthetic images

4.4.2 Multi-class pose estimation

Single class pose estimation can be extended to multiple object classes. In this case, a single class pose estimation is conducted for each of the object classes, which gives a list of proposed poses for each class. The scores which are used to select the best candidates within each class can be compared across the classes, thus giving an indication to which class the unknown object is likely to belong to. The rationale is that we are more likely to find a matching set of views of the correct object class than of an incorrect one.

For the multi-class pose estimation tests, four objects were used (TRUCK, BLOCKCYL, PEGBLOCK, L_SHAPE), and 25 images were generated from random viewpoints for each of the objects. These 100 test images were submitted to single class pose estimation against

each of the object using the models from Section 4.4.1. From each pose estimation the score of the single best pose was kept, and the highest score for an unknown image was used to propose an object class for that image.

Table 4.19 shows part of the results of the experiment. The tendency is that the highest best is significantly higher then the rest of the scores. In fact the proposed object class was correct in all 100 cases, which is not so surprising considering the accuracy of the single class pose estimation from synthetic images.

index	l_shape	pegblock	blockcyl	truck	proposal
+0.068+0.511-0.857	1000140	**2000294**	1000092	1700006	pegblock
+0.078+0.992+0.100	3000700	4001726	3000006	**8002004**	truck
+0.107+0.653+0.750	**4001189**	1000126	2800934	2600008	l_shape
+0.141-0.914-0.382	2000528	4001726	3000006	**9000452**	truck
+0.163-0.969+0.188	3000700	4001726	3000006	**8002149**	truck
+0.193-0.634+0.749	4000910	**5004399**	2000002	2900143	pegblock
+0.204+0.961+0.187	3000700	4001726	3000006	**8002536**	truck
+0.238-0.152+0.959	2000528	4001726	3000006	**7000191**	truck
+0.249-0.754+0.609	2000597	2000072	**6006905**	4700065	blockcyl
+0.255-0.702-0.664	2000780	**4002648**	2000136	2800124	pegblock
+0.270+0.040-0.962	1000284	**2000286**	1700005	1600002	pegblock
+0.305-0.321+0.897	4000910	**5004207**	2000002	2900143	pegblock
+0.313+0.364-0.877	**5004988**	2001158	3000187	2900287	l_shape
+0.332-0.021-0.943	1000282	900036	**2000208**	1300002	blockcyl
+0.366+0.731+0.575	**5004988**	2001158	3000517	2900287	l_shape
+0.406-0.319+0.856	**4002854**	2000332	1000012	2500004	l_shape
+0.427+0.662-0.616	**5004988**	2001158	3000187	2900287	l_shape
+0.446+0.791-0.419	1000728	2000122	**4003477**	3500021	blockcyl
+0.447+0.631-0.634	2000528	4001726	3000964	**10000674**	truck
+0.452-0.877+0.163	3000700	3001615	3000006	**7000333**	truck

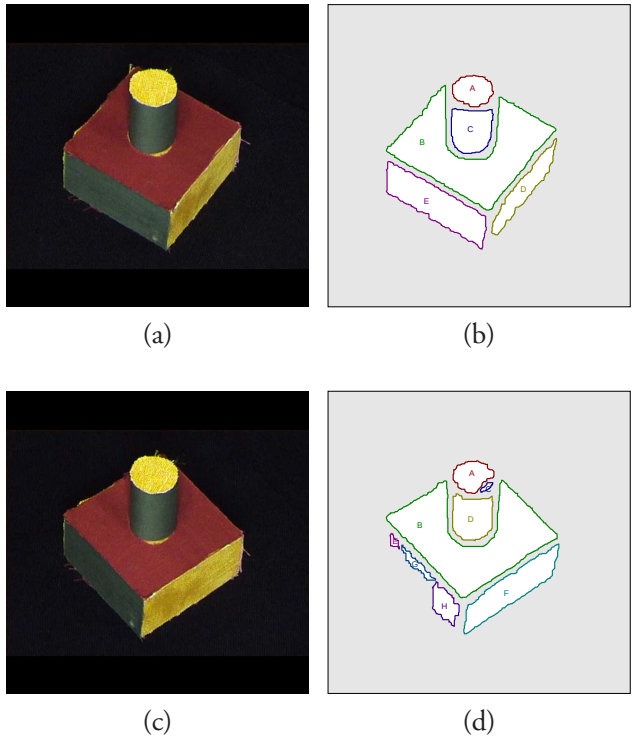
Table 4.19: Multi-class pose estimation results

4.5 Pose estimation from real images

Pose estimation from real image data is much more of a challenge than from synthetic data. Even in a controlled environment, imperfections of the camera, the lighting or the object itself impose increased difficulties for feature extraction. Separating the object from the background is not so trivial as in ray traced images with perfect black background. Also, parame-

ters for image segmentation must be chosen carefully so over-, or under-segmentation do not occur.

In the test setup the objects were placed on a black background and the images were taken with an RGB camera in 640x480 pixels resolution, the light source aligned with the camera angle and placed as close to the lens as possible. The images were padded with black stripes to 640x640 and scaled to 512x512 to match the dimension of the synthetic images. No further scaling, rotation or other adjustments were applied. The high level features were extracted using the same set of tools as for ray traced data, although with a different set of parameters tuned for optimal segmentation of these images.



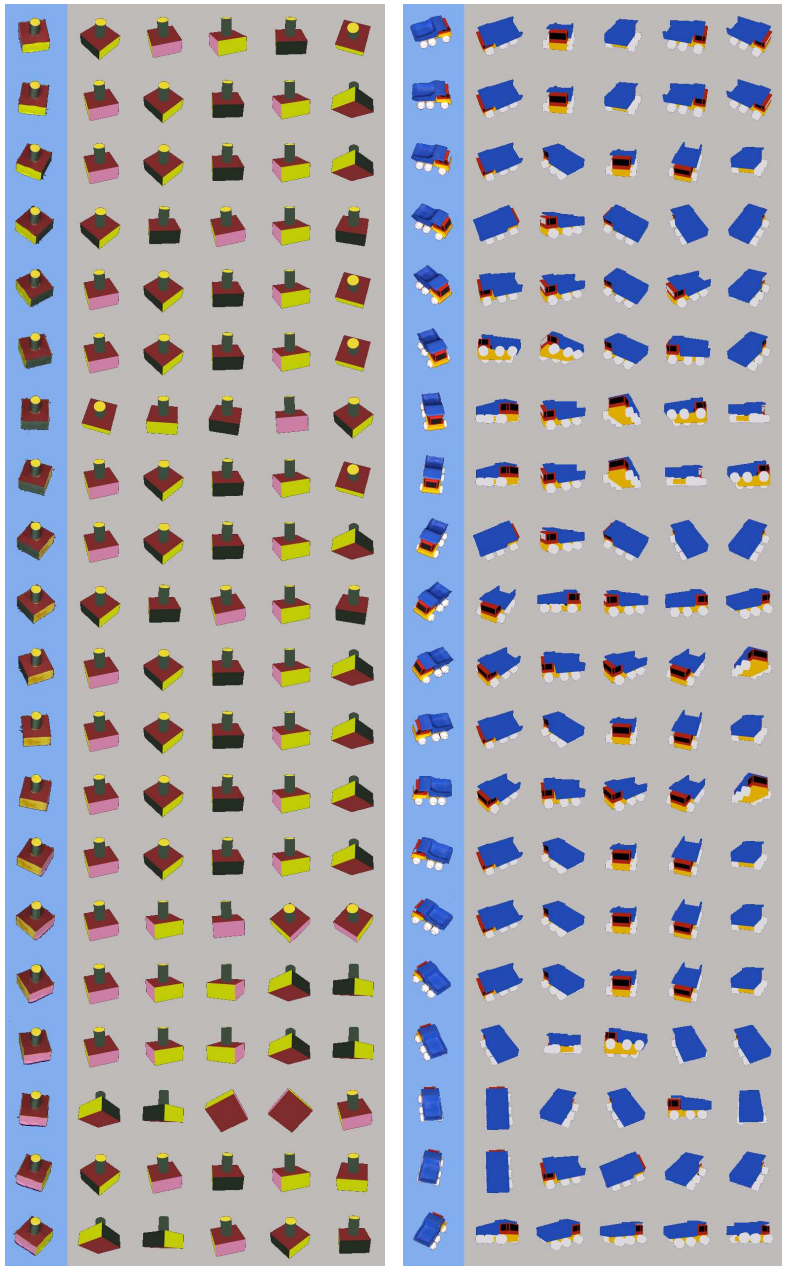
(a) (b)

(c) (d)

Figure 4.24: Correct segmentation (top row, (a)input image and (b)contours) and incorrect segmentation (bottom row, (c)input image and (d)contours), where several contours are split, (A+C and E+G+H).

The lack of exact viewing coordinates made automated judgements of the results less feasible, so the evaluations were done manually based on visual similarity. Figure 4.25 shows a set of

test images and the first five suggested poses of the PEGBLOCK(a) and the TRUCK(b). As expected the precision is not quite as good as for synthetic images, yet if we disregard rotation and mirroring, many of the shortlists contain at least one fairly good candidate. A closer study of the test images which did not have an acceptable candidate on the shortlist indicates that these were likely to have segmentation errors resulting in bogus regions. Figure 4.24 shows such an image ((c) and (d)), compared to a correctly segmented image ((a) and (b)). The overall assessment of the results is that the method performs reasonably well on correctly segmented images, but it is sensitive to the errors in feature extraction.



(a) (b)

Figure 4.25: Results of pose estimation on real images of the PEGBLOCK(a) and the TRUCK(b). For each unknown pose (blue background) the first five suggestions are shown.

Chapter 5

Conclusions

5.1 Summary of work

A method for approximating the aspect or appearance graph model of rigid objects, and an application of this model for pose estimation is presented in this thesis. Section 3.3 describes the method for tessellating the viewing sphere with adaptive resolution control. The acquisition device or devices are simulated in order to obtain synthetic observations of the object from different viewing coordinates. The observations are converted to an attributed graph representation and compared to each other using graph matching. The patches of the viewing sphere (initially the 20 faces of a regular icosahedron) are subdivided until the match between the observations from the corners is accepted, or a desired resolution is reached.

In Section 3.4 the aspects of the objects are determined by grouping together the patches with similar observations. The clustering process uses the graph matching data available from the previous stage. The views belonging to each cluster are indexed into reference classes using the feature data extracted from the images, which is also carried on from the first stage.

The resulting model is applied in a pose estimation algorithm (Section 3.5) using the accumulated and indexed information from the collected views of the object over the different aspects. Weighted or simple bipartite matching is applied for matching the unknown pose to the reference classes. The pose estimation algorithm has been tested on various objects using both simulated and real image data.

All the algorithms and tools presented in the thesis have been implemented and integrated

into a recognition system. Chapter 4 gives experimental results of model construction and pose estimation for a range of objects.

5.2 Evaluation of results

Based on the experiments in Chapter 4 we can conclude that generating the aspect graph model has been successful for the selected set of objects. The correctness of the algorithm is also verified by comparing the generated model to the theoretical expectations based on the geometric model.

Execution time for the model generation process varied from a few hours (CUBE, L_SHAPE) to several days (TRUCK) for level 6 models, depending on the complexity of the object (Table 4.1). As this process is executed only once for each object, and not as part of a real-time recognition system, execution times of this magnitude should not pose major issues for the application. In addition, the model generation algorithm is easily parallelisable (partly supported by the current implementation already), which opens for reducing the execution time by using more than one machine (or CPU). Since the overhead of the algorithm in a distributed setup is small, the execution time is expected to be close to inversely proportional to the number of CPU's (within reasonable limits).

The results also demonstrate that despite its complexity, it is possible to use graph matching in practical applications. In the vast majority of cases the heuristics gave fast convergence, although a small percentage of total matching calls did not give acceptable results within the allowed time frame. It is important that the application is prepared to deal with these cases; in the experiments presented in this work they were treated as an unacceptable match with a particularly high cost. Further refinement of the model in these areas will lead to increasing similarity between the images/graphs and matching will be more likely to complete within the time allowed. Also, in most cases when a good match is found at the end, the matching process finishes in much less time.

Once the model is ready, it can efficiently be applied to recognition and pose estimation tasks. The time spent on matching an unknown image to the model classes was comparable to the time spent on preprocessing and extracting features from a single image, typically much less than that (being in the sub-second range). Sorting the reference classes by the percentage of viewing area covered also helps increasing the probability of an early match.

As far as the precision and reliability of the pose estimation algorithm is concerned, we can conclude that it performs well on synthetic images. For most objects the correct pose was

amongst the top five suggestions in over 90-95% of the cases. Even for the model of the truck which is the most complex of the objects, precision at five was over 88%. For real life data, the performance is not quite as good, but still acceptable judged by visual similarity between the image of the unknown pose and the suggestions. (Due to the lack of reliable viewing coordinates for the real images, performing automated tests on real images was not possible.) Test results show that the algorithm is highly dependent on image segmentation and feature extraction, and errors during this process are likely to propagate and jeopardise the success of the pose estimation. The main focus of this thesis being on the model building part, these issues have not been investigated further.

5.3 Indication for further work

A natural step forward is to extend the set of features used for clustering patches and generating reference classes. Currently only contour (region) attributes were used, while the relations between these were ignored. More features would increase the reliability of the matching; however it would also make the process more complex. There is also room for improvement in handling rotational symmetries and scaling.

Applying rough set theory (Pawlak et al. [119], Pawlak [120]) or fuzzy sets (Zadeh [175]) seems to be another plausible approach to describing reference classes and performing matching as presented in Sections 3.4.4 and. 3.5. Due to the variation in the features over different views in the reference classes, it is impossible (or at least very difficult) to define exact boundaries in feature space; hence the indication to use rough sets or fuzzy sets, which are invented to deal with this kind of problems.

Due to the generic nature of the surface subdivision algorithm the underlying object/image features can be replaced by any other set of features. The only requirement is a cost function to be defined which measures the similarity between the different views. The algorithm performs well if the match cost between views belonging to different aspects stays high while between views within the same aspect it decreases as the viewing coordinates are getting closer.

For the current subdivision algorithm, the viewing space is been restricted to the surface of a viewing sphere. This is not considered as a serious limitation, as the camera to object distance is usually many times larger than the extent of the object, so the aspects do not change significantly with variations in the distance except for scaling. However, for approximating the 3D aspect graph or for dealing with situations when the camera may be placed very near the object, the possibilities of extending the algorithm to three dimensions could be investigated.

Appendix A

Implementation details

A.1 Software

In the following a brief overview of some the main components of the modelling system is given. The intention is not to provide a complete users manual for the software as that would exceed the limits of this work, but to give an insight into various parts of the system and how these are connected to each other.

This list is limited to the key components, and the description of the most important parameters and options.

A.1.1 dyn

usage:

```
dyn -l n [-g] [-t thresh1] [-u thresh2] object_name
options:  -l   icosahedron subdivision level
          -t   graph matcher threshold
          -u   graph matcher threshold, if different (higher) for partitioning phase
          -g   stop after graph generation (no partitioning)
```

description:
Generates the view sphere partitioning from the object geometry and surface properties. Input is the description of the object which is used by the ray tracer, and a parameter file for the

graph matcher.

The object is centred in $[0, 0, 0]$, scaling and camera angle are set corresponding to real test environment, or selected such that the object is entirely visible from the unit sphere. Image resolution is at 512x512.

Prior to starting `dyn`, the ray tracer, feature extractor and graph matcher servers (`bobd`, `featserv` and `graphserv`) must be started. (These may be running on separate machines on the network).

implementation:
C++, uses the LEDA libraries and Berkeley DB

A.1.2 bobd

usage:
```
bobd -S port object_name
options:  -S   port where the server listens
```

description:
An extended version of the Bob ray tracer [163], with support for client/server operation. In server mode, `bobd` reads the object description, then accepts connection from client, which send studio parameters (viewing coordinates, camera angle, etc.) and receive the ray traced image.

implementation:
C

A.1.3 featserv

usage:
```
featserv port
options:  none
```

description:
A client/server wrapper for the image processing and feature extraction pipeline [23].

implementation:
C (tools: C, Pascal, Fortran, Perl, shell script).

A.1.4 graphserv

usage:
```
graphserv port
options:  none
```

description:
A client/server wrapper for the graph matcher [19].

implementation:
C (graph matcher: Pascal).

A.1.5 part2details

usage:
```
part2details sgm_db partfile [detfile]
options:  none
```

description:
Fill region details from SGM (attributed relational graph) database into the partition file generated by `dyn`.

implementation:
Perl script

A.1.6 canon.pl

usage:
```
canon.pl mch_db <det_input >canonical_det_output
options:  none
```

description:
Canonical renaming of the regions, based on graph matching results (from the MCH database).

implementation:
Perl script

A.1.7 det2attr

usage:
```
det2attr [detfile [attrfile]]
options:  none
```

description:
Extracts the list of attribute vectors for all regions.

implementation:
Perl script

A.1.8 km_nodes

usage:
```
km_nodes nodes #clust means [[class] weights]
options:  -h  help
```

description:
Build k-means clusters based on the attribute vectors. Reads the vectors from nodes file, and builds #clust clusters. The mean vectors for each cluster are written to the means file, and the class file – if specified – gives a classification of each vector (with optional weights).

implementation:
C++, uses LEDA.

A.1.9 findclass, findclass_weighted.pl

usage:
```
findclass unknown reference num_candidates >index
options:  none
```

description:
For a list of unknown poses and a list of reference classes finds the best match for each pose. num_candidates it the number of possible poses to propose, using binary or weighted scoring.

implementation:
C++ (uses LEDA), and Perl script.

A.2 File formats

A.2.1 .b, .bo, .bc, .img, OBJECT_img.db

Scene, object, colour and image Files for the `bob` ray tracer. Detailed format of these files can be found in [163]. The **.img** files generated during model building are stored in a database.

blockcyl.bo

```
#define COLOR1 0.237 0.105 0.878
#define COLOR2 0.653 0.392 0.572
...
surface {
  diffuse COLOR1
}
polygon {
  points 4
  vertex -2.000000 1.000000 1.000000
  vertex -2.000000 1.000000 -1.000000
  vertex -2.000000 -1.000000 -1.000000
  vertex -2.000000 -1.000000 1.000000
}
...
```

A.2.2 .sgm, OBJECT_sgm.db

Attributed relational graph description format, and database containing the graphs generated for the model.

blockcyl+0.222-0.842+0.491.sgm

```
1 1 1
[A[@id=cftsrv;ext:sqr:aa(5770):mm(19501,153,16,0):cg(255,377):      \
  cr(63,158,255)]][[8i][@rep= 1 1]][B[@id=cftsrv;ext:tri:aa(17180): \
  mm(18086,0,128109,3523):cg(255,299):cr(158,63,255)]]
1 2 1
[A[@id=cftsrv;ext:sqr:aa(5770):mm(19501,153,16,0):cg(255,377):      \
  cr(63,158,255)]][[a][@rep= 1 ]][C[@id=cftsrv;ext:tri:aa(3812):    \
  mm(15987,6,191882,1771):cg(162,255):cr(255,63,255)]]
1 3 1
[A[@id=cftsrv;ext:sqr:aa(5770):mm(19501,153,16,0):cg(255,377):      \
  cr(63,158,255)]][[7][@rep= 1 ]][D[@id=cftsrv;ext:tri:aa(5022):    \
  mm(15547,53,64298,144):cg(339,235):cr(255,63,255)]]
...
```

A.2.3 .red, .grn, .blu

The three colour channels of an image, in REB image format (raw 8-bit data with a header).

A.2.4 .cnt, .chn8, OBJECT_chn.db

Contour and chain code files. The **.cnt** file is a list of contour points of the format <contour_number x y 16-way_direction_code>. <0 0 0> marks the end of the list. **.chn8** files conatain 8-way chain codes, one line per contour. The format is <start_x start_y chain code>. The chain code is a list of <[rep]dir> elements, where dir is one of the A-H direction code letters and rep is and optional number specifying how many times the code is repeated. <-1 -1> marks the end of the list. The **.chn8** format contours are stored in a database.

blockcyl+0.222-0.842+0.491.chn8

```
253 404 5AH15AH6AH5AH4AH3AH2AH2AH2AH3AHAHA3HA2HA3HG2H2GH3GFGFG5FE2FE2FEF \
        2EFEF2EF2EF2EF3EF3EF4EF6EF10EF21ED10ED6ED4ED3ED3ED2ED2ED2EDED2ED \
        E2DE2DE5DCDCD3CB2C2BC3BA2BA3BABAB3AB2AB2AB2AB3AB4AB5AB6AB15A
185 368 A4HA2HA2HAH2AHAH2AH2AH2AH3AH3AH4AH6AH10AH25AB10AB6AB4AB3AB3AB2AB \
        2AB2ABAB2ABA2BA2BA4BAGF24GF24GF25GF24GF15GF2GFGFG7FE3FEFEFEFEFEF \
        2EF2EF2EF3EF3EF5EF7EF21ED7ED5ED3ED3ED2ED2ED2EDEDEDEDEDE3DE7DCDCD \
        2CD15CD24CD25CD24CD24CD
...
-1 -1
```

A.2.5 .vert

`dyn` stores the vertex data in these files. The first line of the file gives the number of vertices listed, then there is one line for each vertex, in the format `<3 x y z id nr>`. `3` is the number of coordinates (dimensions), `x,y,z` are the vertex coordinates, `id` is a unique ID, and `nr` is the number of regions in the image generated form the viewpoint.

blockcyl.6.vert

```
18039
3  -1.0000000000  0.0000000000  0.0000000000 82 2
3  -0.9998072406 -0.0196336858 -0.0000000713 12743 2
3  -0.9998072406  0.0196336858 -0.0000000713 12805 2
3  -0.9997919713 -0.0103355404  0.0175838161 12744 2
3  -0.9997919713  0.0103355404  0.0175838161 12804 2
3  -0.9993814181  0.0000000000  0.0351678988 12891 4
...
```

A.2.6 .part

Partition file generated by `dyn`. The format is:

```
Partition 1
        <patch>
        <patch>
        ...
Partition 2
        <patch>
        ...
...
```

Each <patch> specifies a triangular patch, and has the following elements: <vect id nr vect id nr vect id nr level>. <vect> is a vector determining a point on the viewing sphere in the format <3(=number of coordinates) x y z>, <id> is a vertex ID, <nr> is the number of regions visible from that vertex. <level> is the level of subdivision at which the triangle was generated.

blockcyl.6.part

```
Partition 1
3 -0.951 -0.309 -0.000 23 3 3 -0.988 -0.156 -0.000 307 3     \
    3 -0.964 -0.237 -0.117 308 3 3
3 -0.951 -0.309 -0.000 23 3 3 -0.954 -0.301 0.018 12434 3    \
    3 -0.957 -0.290 -0.000 1 2675 3 6
3 -0.957 -0.290 -0.000 12675 3 3 -0.959 -0.282 0.018 12676 3 \
    3 -0.962 -0.271 -0.000 4688 3 6
3 -0.962 -0.271 -0.000 4688 3 3 -0.965 -0.263 0.018 12679 3  \
    3 -0.968 -0.252 -0.000 12680 3 6
...
```

A.2.7 .det

The partition details, extracted from the **.part** files with `part2det`. This format is used by `remintdet`, `canon` and `det2attr`. The file structure is the following:

```
<total number of groups>
        <number of graphs in this group>
        <serial number of this group>
                uniq_id uniq_id theta phi
                <number of nodes for graph> <view index>
                <graph node in .sgm format>
                <graph node in .sgm format>
```

```
        ...
        uniq_id uniq_id theta phi
        <number of nodes for graph> <view index>
        <graph node in .sgm format>
        ...
<number of graphs in this group>
<serial number of this group>
        ...
...
```

blockcyl.6.det

```
88
238
1
1 1 -155 -79
3 -0.163-0.984-0.074
[A[@id=na;ext:sqr:aa(17959):mm(21021,0,11183,777):cg(255,326):cr(158,63,255)]]
[B[@id=na;ext:sqr:aa(40367):mm(20618,96,4008,92):cg(262,187):cr(255,63,63)]]
[C[@id=na;ext:sqr:aa(151):mm(20649,426,0,0):cg(119,195):cr(42,42,255)]]
2 2 -164 -80
3 -0.164-0.985-0.044
[A[@id=na;ext:sqr:aa(18212):mm(21027,0,6073,592):cg(255,325):cr(158,63,255)]]
[B[@id=na;ext:sqr:aa(40180):mm(20612,96,3563,51):cg(262,185):cr(255,63,63)]]
[C[@id=na;ext:sqr:aa(88):mm(20854,431,0,0):cg(120,229):cr(63,63,255)]]
3 3 173 -80
3 -0.165-0.986+0.018
[A[@id=na;ext:sqr:aa(18493):mm(20989,0,1717,152):cg(255,323):cr(158,63,255)]]
[B[@id=na;ext:sqr:aa(160):mm(20840,433,0,0):cg(120,212):cr(63,63,255)]]
[C[@id=na;ext:sqr:aa(39705):mm(20643,97,3356,76):cg(262,181):cr(255,63,63)]]
...
```

A.2.8 .attr

Attribute vectors extracted from the partition details (**.det**) files.

```
<number of regions> <number of graphs>
        <partition id> <graph id> <region id> <attribute vector>
        <partition id> <graph id> <region id> <attribute vector>
        ...
```

blockcyl.attr

```
80515 17060
1 1 0  17959 21021 0 11183 777 255 326 158 63 255
1 1 1  151 20649 426 0 0 119 195 42 42 255
1 1 2  40367 20618 96 4008 92 262 187 255 63 63
1 2 0  18212 21027 0 6073 592 255 325 158 63 255
1 2 1  88 20854 431 0 0 120 229 63 63 255
1 2 2  40180 20612 96 3563 51 262 185 255 63 63
1 3 0  18493 20989 0 1717 152 255 323 158 63 255
1 3 1  160 20840 433 0 0 120 212 63 63 255
1 3 2  39705 20643 97 3356 76 262 181 255 63 63
...
88 4 0  2322 18256 231 53088 4251 296 335 158 63 255
88 4 1  11691 18197 98 9268 91 188 306 63 63 255
88 4 2  208 18016 295 1779 413 231 374 158 63 255
88 4 3  8946 18596 302 3173 369 315 237 255 63 63
88 4 4  30903 18180 67 54713 1329 250 170 63 255 255
```

A.2.9 .means

Mean vectors of the clusters from k-means clustering.

blockcyl.means

```
30
9853 19141 159 4508 193 253 189 62 254 63
9870 19144 158 4585 197 254 188 62 62 255
17330 19018 204 15785 724 255 194 254 254 63
121 19960 396 1 0 250 193 219 53 89
95 19478 378 0 0 252 198 43 43 255
176 18766 350 2 0 253 200 58 240 73
...
```

A.2.10 .class

Description of the equivalence classes in terms of the region types. The format of the file is the following:

```
<class id> <number of regions>
        <num region types> <total num graphs> <type id>:<num graphs> ...
        <num region types> <total num graphs> <type id>:<num graphs> ...
        ...
<class id> <number of regions>
```

```
        <num region types> <total num graphs> <type id>:<num graphs> ...
        ...
...
```

blockcyl.class

```
1 3
    2 238   22:222 29:16
    2 238   1:237 4:1
    1 238   16:238
2 3
    2 236   22:220 29:16
    2 236   1:235 4:1
    1 236   2:236
3 2
    1 92    22:92
    1 92    2:92
4 3
    1 261   22:261
    1 261   2:261
    1 261   0:261
...
```

A.2.11 .index

Results of the pose estimation, generated by `findclass` or `findclass_weighted.pl`

```
<unknown image name>
        <proposed pose> <score> <list of region matches> ...
        <proposed pose> <score> <list of region matches> ...
        ...
<unknown image name>
...
```

test1.index

```
+0.015+0.442+0.897
38      5001664 0-0(482) 2-2(207) 3-3(90) 1-1(237) 4-4(648)
33      4902646 0-0(941) 2-3(71) 4-4(1390) 1-2(170) 3-1(74)
39      4902154 0-0(909) 4-3(1046) 2-4(2) 1-2(119) 3-1(78)
44      4000898 0-0(433) 3-2(54) 1-1(168) 2-3(243)
36      3901220 0-0(933) 1-2(168) 2-4(42) 3-1(77)
45      3901203 0-0(893) 2-3(67) 1-2(165) 3-1(78)
...
```

A.3 Driving the modelling system

1. Create the object model in `.b` format for the ray tracer. (The object should be centred in $[0, 0, 0]$, and scaling, view angle and projection should be set such that the whole object is visible when the viewpoint is on the unit sphere. The

   ```
   from x y z
   ```

   element may be left out from the studio structure of the `.b` file, it will be overridden each time ray tracing is called as `dyn` takes control over the viewpoints. The `.b` file could look something like this:

   ```
   /* rl.b */
   #include color.bc
   studio {
         at 0 0 0
         up 0 1 0.05
         angle 50
         resolution 128 128
         antialias none
         threshold 1
         aspect 1
         ambient 1 1 1
         }
   #include rl.bo
   transform {scale 0.2 } RL_N transform_pop
   ```

   RL_N is the object definition itself, included from rl.bo:

   ```
   /* rl.bo */
   /* colors */
   #define C1 .682 .631 .345
   #define C2 .682 .631 .345
   ...
   #define C8 .615 .588 .521
   /* vertices */
   #define V1 vertex -2.5 5.7 +1.85
   #define V2 vertex -2.5 0 +1.85
   ```

```
#define V3 vertex +2.5 0 +1.85
...
#define V12 vertex -0.6 5.7 -1.85
/* surfaces */
/* RL L-shape object, in real size (unit: cm) /*
#define RL \
surface { diffuse C1 } polygon { points 6 V1 V2 V3 V4 V5 V6 } \
surface { diffuse C2 } polygon { points 6 V12 V11 V10 V9 V8 V7 } \
surface { diffuse C3 } polygon { points 4 V1 V6 V12 V7 } \
surface { diffuse C4 } polygon { points 4 V1 V7 V8 V2 } \
surface { diffuse C5 } polygon { points 4 V6 V5 V11 V12 } \
surface { diffuse C6 } polygon { points 4 V5 V4 V10 V11 } \
surface { diffuse C7 } polygon { points 4 V4 V3 V9 V10 } \
surface { diffuse C8 } polygon { points 4 V2 V8 V9 V3 }
/* RL_N L-shape object normalized to fit in (±1,±1,±1) cube */
#define RL_N \
transform { translate 0 -2.85 0 scale 0.35088 } RL transform_pop
```

2. Create a directory for the object, and place the relevant files there (`.b`, `.bo`, `.bc`).

3. Start the ray tracing, feature extraction and graph matching servers (`bobd`, `featserv` and `graphserv`) which `dyn` will connect to.

4. Run `dyn` with the name of the object file. This might take a while, depending on the object, the required depth and the computing power the running time can be anything from some minutes to several days. See Table 4.1 for some examples.
   As `dyn` is computing the model, a large number of files will be created. For each view position examined, there will be an image file (`.img`), a contour file (`.chn8`) and an attributed relational graph file (`.sgm`). These files have the view position encoded in the file name. When `dyn` is finished, a vertex file (`.vert`), a graph file (`.graph`), a partition file (`.part`), a postscript image of the partitions (`.ps`), a file containing the number of regions for each view (`.nr`) and a match file (`.mch`) is written. The files are named with the object name and some are indexed with the depth. For detailed description of these files see Appendix A.2.
   For increased efficiency the files are not stored directly in the directory, but are collected into databases instead, one for each file type, which allows for faster lookup and reduces disk space consumption. `dyn` will access the databases directly, other applications can use the `dbxtr` and `dbdump` tools to retrieve some or all files from a database.

Using the sample files mentioned above, the command line invoking `dyn` in the background:

```
sid% nohup dyn -l 3 -v -s rl >& rl.log &
```

5. The next step is to organise the information in the partition file. First, `part2details` is used to convert the extract the partition details. Then, internal contours are removed with `remintdet`. The tool `canon.pl` is applied for canonical renaming of the nodes and for generating a `.map` file which maps the partitions to a representative view. The script `canon.script` will do all this.

6. For the pose estimation, run `det2attr` to extract the attributes from the `.det` file. Then use `km_nodes` to find the clusters (k-means). The script `known.script` will do this.

7. Now we have the classes ready. If we then want to classify a set of unknown images, `unknown.script` can be used. It extracts the attributes from a set of unknown files (`feat2det` and `det2attr`), calls `cl_unk` to find the clusters for the regions, and runs `findclass` to find the most likely classes. The index file contains the suggested classes. At last `index2ps` can be called to create the Postscript output for human reading.

Bibliography

[1] A. M. Abdulkader. *Parallel Algorithms for Labelled Graph Matching*. PhD thesis, Colorado School of Mines, 1998.

[2] Peter Allen. Surface descriptions from vision and touch. In *Proceedings of the 1984 IEEE International Conference on Robotics and Automation*, pages 394–397, March 1984.

[3] A. Ansar and K. Daniilidis. Linear pose estimation from points or lines. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(5):578–589, May 2003.

[4] F. Arman and J. K. Aggarwal. Automatic generation of recognition strategies using CAD models. In *Workshop on Directions in Automated CAD-Based Vision*, pages 124–133, June 1991.

[5] M. Asada, H. Ichikawa, and S. Tsuji. Determining surface orientation by projecting a stripe pattern. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(5):749–754, 1988.

[6] M. Asada and S. Tsuji. Shape from projecting a stripe pattern. In *Proceedings of the 1987 IEEE International Conference on Robotics and Automation*, pages 787–792, March 1987.

[7] Eric Backer and Jan J. Gerbrands. Inexact graph matching used in machine vision. *NATO ASI Series F, Pattern Recognition Theory and Applications*, F30:347–356, 1987.

[8] W. W. R. Ball and H. S. M. Coxeter. *Mathematical Recreations and Essays*. New York: Dover, 13th edition, 1987.

[9] Dana H. Ballard and Chritopher M. Brown. *Computer Vision*. Englewood Cliffs, New Jersey, 1982.

[10] D. A. Basin. A term equality problem equivalent to graph isomorphism. *Information Processing Letters*, 54:61–66, 1994.

[11] Jeffrey S. Beis and David G. Lowe. Learning indexing functions for 3D model-based object recognition. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 275–280, 1994.

[12] M. Bennamoun and B. Boashash. A vision system for automatic object recognition. In *Proceedings of the 1994 IEEE International Conference on Systems, Man, and Cybernetics 'Humans, Information and Technology'*, pages 1369–1374, October 1994.

[13] Robert Bergevin and Mertin D. Levine. Generic object recognition: Building and matching coarse descriptions from line drawings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(1):19–36, January 1993.

[14] P. J. Besl and R. C. Jain. Three-dimensional object recognition. *ACM Computing Surveys*, 17:75–145, 1985.

[15] Bir Bhanu and Thomas C. Henderson. CAGD based 3D vision. In *Proceedings of the IEEE International Conference Robotics and Automation*, pages 411–417, 1985.

[16] Bir Bhanu and Chih-Cheng Ho. CAD-based 3D object representation for robot vision. *IEEE Computer*, 20(8):19–35, August 1987.

[17] T. Binford. Visual perception by computer. In *Proceeding or the IEEE Conference on Systems and Control*, Miami, FL, December 1971.

[18] Richard E. Blake. A matching method for difficult shapes from engineering drawings. In *Proceedings of the 4th Scandinavian Conference on Image Analysis*, pages 67–74, June 1985.

[19] Richard E. Blake. Development of an incremental graph matching device. In *Pattern Recognition Theory and Applications*, volume F30 of *NATO ASI Series*, pages 357–366. Springer-Verlag, 1987.

[20] Richard E. Blake. The use of scott's lattice theory as a basis for combining items of evidence. *Pattern Recognition Letters*, 7(3):151–155, March 1988.

[21] Richard E. Blake. A partial ordering for relational graphs applicable to varying levels of detail. *Pattern Recognition Letters*, 11(5):305–312, May 1990.

[22] Richard E. Blake. Partitioning graph matching with constraints. *Pattern Recognition*, 27(3):439–446, March 1994.

[23] Richard E. Blake and Peter Boros. The extraction of structural features for use in computer vision. In *Proceedings of the 2nd Asian Conference on Computer Vision*, pages 583–587, 1995.

[24] Robert C. Bolles and Patrice Horaud. 3DPO: A three dimensional part orientation system. *International Journal of Robotics Research, 5:3-26, 1986*, (5):3–26, 1986.

[25] Robert C. Bolles, Patrice Horaud, and Marsha Jo Hannah. 3DPO: A three dimensional part orientation system. In *Proceedings IJCAI*, pages 1116–1120, 1983.

[26] K. Bowyer, J. Stewman, L. Stark, and D. Eggert. ERRORS-2: a 3D object recognition system using aspect graphs. In *Proceedings of the 9th International Conference on Pattern Recognition*, pages 6–10, November 1988.

[27] Kevin Bowyer, Maha Sallam, David Eggert, and John Stewman. Computing the generalized aspect graph for objects with moving parts. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 15(6):605–610, June 1993.

[28] Kevin W. Bowyer and Charles R. Dyer. Aspect graphs: An introduction and survey of recent results. *Int'l Journal on Imaging Systems and Technology*, 2:315–328, 1990.

[29] Kevin W. Bowyer, David Eggert, John Stewman, and Louise Stark. Developing the aspect graph representation for use in image understanding. In Hatem Nasr, editor, *Selected Papers on Model-Based Vision (Milestone Series # 72)*, pages 198–216. SPIE Press, 1993.

[30] Rodney A. Brooks. Model based three dimensional interpretation of two dimensional images. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 360–370, August 1981.

[31] H. Bunke. Error correcting graph matching: on the influence of the underlying cost function. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(9):917–922, September 1999.

[32] G. Castore. Solid modeling, aspect graphs and robot vision. In Pickett and Boyse, editors, *Solid Modeling by Computer*, pages 277–292. New York: Plenum Press, 1984.

[33] Glen Castore and Carol Crawford. From solid model to robot vision. In *Proceedings of the 1984 IEEE International Conference on Robotics and Automation*, pages 90–92, March 1984.

[34] Chin-Chun Chang and Wen-Hsiang Tsai. Reliable determination of object pose from line features by hypothesis testing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(11):1235–1241, November 1999.

[35] J.-L. Chen and G. C. Stockman. Determining pose of 3D objects with curved surfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(1):52–57, January 1996.

[36] J.-L. Chen and G. C. Stockman. Indexing to 3D model aspects using 2D contour features. In *Proceedings of the 1996 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 913–920, June 1996.

[37] S. Chen and H. Freeman. Computing characteristic views of quadric-surfaced solids. In *Proceedings of the 10th International Conference on Pattern Recognition*, pages 77–82, June 1990.

[38] S. Chen and H. Freeman. On the characteristic views of quadric-surfaced solids. In *IEEE Workshop on Directions in Automated CAD-Based Vision*, pages 34–43, June 1991.

[39] S.-W. Chen and G. Stockman. Object wings-2 1/2 D primitives for 3D recognition. In *Proceedings of the 1989 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 535–540, June 1989.

[40] S.-W. Chen and G. Stockman. Wing representation for rigid 3d objects. In *Proceedings of the 10th International Conference on Pattern Recognition*, pages 398–402, June 1990.

[41] Roland T. Chin and Charles R. Dyer. Model-based recognition in robot vision. In *Computing Surveys*, pages 67–108, 1986.

[42] C. M. Cyr and B. B. Kimia. 3D object recognition using shape similiarity-based aspect graph. In *Proceedings of the 8th IEEE International Conference on Computer Vision*, pages 254–261, July 2001.

[43] E. R. Davies. *Machine Vision: Theory, Algorithm, Practicalities*. Academic Press, San Diego, 2nd edition, 1997.

[44] Daniel DeMenthon and Larry S. Davis. Exact and approximate solutions of the perspective-three-point problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(11):1100–1105, November 1992.

[45] S. J. Dickinson, P. Jasiobedzki, G. Olofsson, and H. I. Christensen. Qualitative tracking of 3-d objects using active contour networks. In *Proceedings of the 1994 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 812–817, June 1994.

[46] S. J. Dickinson, A. P. Pentland, and A. Rosenfeld. Qualitative 3-D shape reconstruction using distributed aspect graph matching. In *Proceedings of the Third International Conference on Computer Vision*, pages 257–262, December 1990.

[47] S. J. Dickinson, A. P. Pentland, and A. Rosenfeld. 3-D shape recovery using distributed aspect matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):174–198, February 1992.

[48] C. Dorai, G. Wang, A. K. Jain, and C. Mercer. From images to models: automatic 3D object model construction from multiple views. In *Proceedings of the 13th International Conference on Pattern Recognition*, pages 770–774, August 1996.

[49] T. Van Effelterre, L. Van Gool, and A. Oosterlinck. Visual recognition of CAD objects with aspect graphs. In *Proceedings of the 1992 IEEE International Symposium on Intelligent Control*, pages 54–59, August 1992.

[50] David W. Eggert. *Aspect Graphs of Solids of Revolution*. PhD thesis, University of South Florida, December 1991.

[51] David W. Eggert and Kevin Bowyer. Computing the perspective projection aspect graphs of solids of revolution. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(2):109–128, February 1993.

[52] David W. Eggert, Kevin Bowyer, Charles R. Dyer, Henrik I. Christensen, and Dmitry B. Goldgof. The scale space aspect grap. In *Proceedings of the 1992 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 335–340, June 1992.

[53] David W. Eggert, Kevin Bowyer, Charles R. Dyer, Henrik I. Christensen, and Dmitry B. Goldgof. The scale space aspect graph. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11):1114–1130, November 1993.

[54] David W. Eggert and Kevin W. Bowyer. Computing the orthographic projection aspect graph of solids of revolution. In *Proceedings of IEEE Workshop on Interpretation of 3D Scenes*, pages 102–108, November 1989.

[55] David W. Eggert and Kevin W. Bowyer. Perspective projection aspect graph of solids of revolution: An implementation. In *IEEE Workshop on Directions in Automated CAD-Based Vision*, pages 44–53, June 1991.

[56] S. Ekvall, F. Hoffmann, and D. Kragic. Object recognition and pose estimation for robotic manipulation using color cooccurrence histograms. In *Proceedings of the 2003*

IEEE/RSJ International Conference on Intelligent Robots and Systems, volume 2, pages 1284–1289, October 2003.

[57] Ting-Jun Fan, Gerard Medioni, and Ramakant Nevatia. Recognizing 3D objects using surface descriptions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(11):1140–1157, November 1989.

[58] O. Faugeras, J. Mundy, N. Ahuja, C. Dyer, A. Pentland, R. Jain, K. Ikeuchi, and K. Bowyer. Why aspect graphs are not (yet) practical for computer vision. In *Workshop on Directions in Automated CAD-Based Vision*, pages 97–104, June 1991.

[59] A. M. Finch, R. C. Wilson, and E. R. Hancock. Relational matching with mean field annealing. In *Proceedings of the 13th International Conference on Pattern Recognition*, pages 359–363, August 1996.

[60] Leila De Floriani. Feature extraction from boundary models of three-dimensional objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(8):785–798, August 1989.

[61] P. J. Flynn and A. K. Jain. BONSAI: 3D object recognition using constrained search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(10):1066–1075, October 1991.

[62] Patrick J. Flynn and Anil K. Jain. CAD-based computer vision: From CAD models to relational graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(2):114–132, February 1991.

[63] David Forsyth, Joseph L. Mundy, Andrew Zissermann, Chris Coelho, Aaron Heller, and Charles Rothwell. Invariant descriptors for 3D object recognition and pose. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(10):971–991, October 1991.

[64] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New-York, 1979.

[65] Ziv Gigus, John Canny, and Raimund Seidel. Efficiently computing and representing aspect graphs of polyhedral objects. In *Proceedings of the 2nd International Conference on Computer Vision*, pages 30–39, 1988.

[66] Ziv Gigus, John Canny, and Raimund Seidel. Efficiently computing and representing aspect graphs of polyhedral objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(6):542–551, June 1991.

[67] Ziv Gigus and Jitendra Malik. Computing the aspect graph for line drawings of polyhedral objects. In *Proceedings of the 1988 Computer Vision and Pattern Recognition Conference*, pages 654–661, 1988.

[68] Ziv Gigus and Jitendra Malik. Computing the aspect graph for line drawings of polyhedral objects. In *Proceedings of the 1988 IEEE International Conference on Robotics and Automation*, pages 1560–1566, 1988.

[69] Ziv Gigus and Jitendra Malik. Computing the aspect graph for line drawings of polyhedral objects. *IEEE Transactons on Pattern Analysis and Machine Intelligence*, 12(2):113–122, February 1990.

[70] Steven Gold and Anand Rangarajan. A graduated assignment algorithm for graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(4):377–388, April 1996.

[71] Ian E. Gordon. *Theories of Visual Perception.* John Wiley & Sons, 1989.

[72] R. Hanek, N. Navab, and M. Appel. Yet another method for pose estimation: A probabilistic approach using points, lines, and cylinders. In *Proceedings of the 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, page 550, June 1999.

[73] Charles Hansen and Thomas C. Henderson. CAGD-based computer vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(11):1181–1193, November 1989.

[74] R. M. Haralick, H. Joo, C. Lee, X. Zhuang, V.G. Vaidya, and M. B. Kim. Pose estimation from corresponding point data. *IEEE Trans. on Systems, Man and Cybernetics*, 19(6):1426–1446, 1989.

[75] R. M. Haralick and Hyonam Joo. 2D-3D pose estimation. In *Proceedings of the 9th International Conference on Pattern Recognition*, pages 385–391, November 1988.

[76] Robert M. Haralick and Linda G. Shapiro. *Computer and Robot Vision.* Addison-Wesley, Reading MA, 1993.

[77] S. Hati and S. Sengupta. Pose estimation in automated visual inspection using ANN. In *Proceedings of the 1998 IEEE International Conference on Systems, Man, and Cybernetics*, pages 1732–1737, October 1998.

[78] U. Hillenbrand and G. Hirzinger. Object recognition and pose estimation from 3D-geometric relations. In *Proceedings of the 4th International Conference on Knowledge-Based Intelligent Engineering Systems and Allied Technologies*, pages 113–116, 2000.

[79] R. Hoffman and H. R. Keshavan. Evidence-based object recognition and pose estimation. In *Proceedings of the 1987 IEEE International Conference on Systems, Man and Cybernetics*, pages 173–178, November 1989.

[80] R. Hoffman, H. R. Keshavan, and F. Towfiq. CAD-driven machine vision. *IEEE Trans. on Systems, Man and Cybernetics*, 19(6):1477–1488, 1989.

[81] T. Hogg, D. Rees, and H. Talhami. Three-dimensional pose from two-dimensional images: a novel approach using synergetic networks. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, pages 1140–1144, 1995.

[82] J. E. Hopcroft and J. K. Wong. Linear time algorithm for isomorphism of planar graphs. In *Sixth ACM Symposium on Theory of Computing*, 1974.

[83] John E. Hopcroft and Richard M. Karp. An $n^{\frac{5}{2}}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, December 1973.

[84] Berthold K. P. Horn. *Robot Vision*. MIT Press, Cambridge MA, 1986.

[85] M. K. Hu. Visual pattern recognition by moment invariants. *IRE Transactions on Information Theory*, IT-8:179–187, 1962.

[86] H. V. Jagadish and L. O'Gorman. An object model for image recognition. *IEEE Computer*, 22(12):33–41, December 1989.

[87] Ramesh Jain, Rangachar Kasturi, and Brian G. Schunck. *Machine Vision*. McGraw-Hill, New York, 1995.

[88] S. Kameyama and T. Nagata. Generating an aspect graph by set operations on sets of viewpoints. In *Proceedings of the 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1889–1896, July 1993.

[89] A. Kaufman, D. Cohen, and R. Yagel. Volume graphics. *Computer*, 26(7):51–64, July 1993.

[90] A. Khotanzad and J. J.-H. Liou. Recognition and pose estimation of unoccluded three-dimensional objects from a two-dimensional perspective view by banks of neural networks. *IEEE Transactions on Neural Networks*, 7(4):897–906, July 1996.

[91] Gen-ichiro Kinoshita, Eiichi Mutoh, and Kazuo Tanie. Haptic aspect graph representation of 3D object shapes. In *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, pages 1648–1653, May 1992.

[92] Gen-ichiro Kinoshita, Eiichi Mutoh, and Kazuo Tanie. Haptic aspect graph representation of 3D solid object shapes by tactile sensing. In *Proceedings of the 1992 lEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 1912–1917, July 1992.

[93] J. J. Koenderink and A. J. van Doorn. The internal representation of solid shape with respect to vision. *Biological Cybernetics*, 32:211–216, 1979.

[94] D. J. Kriegman and J. Ponce. Computing exact aspect graphs of curved objects: solids of revolution. In *Proc. of IEEE Workshop on Interpretation of 3D Scenes*, pages 116–122, 1989.

[95] D. J. Kriegman and J. Ponce. Computing exact aspect graphs of curved objects: Solids of revolution. *Int'l. Journal on Computer Vision*, 5(2):119–135, 1990.

[96] A. Laurentini. Comments on "efficiently computing and representing aspect graphs of polyhedral objects". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18:57–58, January 1996.

[97] J. Llados, E. Marti, and J. J. Villanueva. Symbol recognition by error-tolerant subgraph matching between region adjacency graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(10):1137–1143, October 2001.

[98] Bin Luo and E. R. Hancock. Structural graph matching using the EM algorithm and singular value decomposition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(10):1120–1136, October 2001.

[99] David Marr. *Vision*. W. H. Freeman, 1982.

[100] G. Marsaglia. Choosing a point from the surface of a sphere. In *Ann. Math. Stat.*, volume 43, pages 645–646. 1972.

[101] David W. Matula. Subtree isomorphism in $O(n^{\frac{5}{2}})$. *Annals of Discrete Mathematics*, 2:91–106, 1978. North-Holland Publishing Company.

[102] B. T. Messmer and H. Bunke. A new algorithm for error-tolerant subgraph isomorphism detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5):493–504, May 1998.

[103] H. Murase and S. K. Nayar. Learning and recognition of 3D objects from appearance. In *Proceedings of IEEE Workshop on Qualitative Vision*, pages 39–50, June 1993.

[104] H. Murase and S. K. Nayar. Learning by a generation approach to appearance-based object recognition. In *Proceedings of the 13th International Conference on Pattern Recognition*, pages 24–29, August 1996.

[105] R. Myers, R. C. Wilson, and E. R. Hancock. Efficient relational matching with local edit distance. In *Proceedings of the 14th International Conference on Pattern Recognition*, pages 1711–1714, August 1998.

[106] R. Myers, R. C. Wilson, and E. R. Hancock. Bayesian graph edit distance. In *Proceedings of the 1999 International Conference on Image Analysis and Processin*, pages 1166–1171, September 1999.

[107] R. Myers, R. C. Wilson, and E. R. Hancock. Bayesian graph edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(6):628–635, June 2000.

[108] K. Nakano and Y. Watanabe. Robot vision system capable of recognizing machine parts using multistage neural networks. In *Proceedings of the 1992 IEEE International Conference on Systems Engineering*, pages 389–392, September 1992.

[109] Vishvjit Nalwa. *A Guided Tour of Computer Vision*. Addison-Wesley, Reading MA, 1993.

[110] S. K. Nayar, H. Murase, and S. A. Nene. Learning, positioning, and tracking visual appearance. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 3237–3244, May 1994.

[111] S. K. Nayar, S. A. Nene, and H. Murase. Real-time 100 object recognition system. In *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, pages 2321–2325, April 1996.

[112] A. Noble, D. Wilson, and J. Ponce. On computing aspect graphs of smooth shapes from volumetric data. In *Proceedings of the Workshop on Mathematical Methods in Biomedical Image Analysis*, pages 299–308, June 1996.

[113] Y. Nomura, D. Zhang, Y. Sakaida, and S. Fujii. 3-D object pose estimation by shading and edge data fusion – simulating virtual manipulation on mental images. In *Proceedings of the 1996 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 866–871, June 1996.

[114] R. Osada, T. Funkhouser, B. Chazelle, and D. Dobkin. Matching 3D models with shape distributions. In *Proceedings of the SMI 2001 International Conference on Shape Modeling and Applications*, pages 154–166, May 2001.

[115] Sung-Il Pae and Jean Ponce. Toward a scale-space aspect graph: solids of revolution. In *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 196–201, June 1999.

[116] L. H. Pampagnin and M. Devy. 3D object identification based on matchings between a single image and a model. In *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, pages 1580–1587, April 1991.

[117] Soon-Yong Park and M. Subbarao. Pose estimation and integration for complete 3D model reconstruction. In *Proceedings of the 6th IEEE Workshop on Applications of Computer Vision*, pages 143–147, December 2002.

[118] T. Pavlidis. *Algorithms for Graphics and Image Processing*. Computer Science Press, 1982.

[119] Z. Pawlak, S. K. M. Wong, and W. Ziarko. Rough sets: Probabilistic versus deterministic approach. In B. Gaines and J. Boose, editors, *Machine Learning and Uncertain Reasoning*, volume 3 of *Knowledge Based Systems*, pages 227–241. Academic Press, 1990.

[120] Zdislaw Pawlak. *Rough Sets – Theoretical Aspects of Reasoning About Data*. Kluwer Academic Press, 1991.

[121] S. Petitjean, J. Ponce, and D. J. Kriegman. Computing exact aspect graphs of curved objects: Algebraic surfaces. *Int'l. Journal on Computer Vision*, 9(3):231–255, 1992.

[122] Euripides G. M. Petrakis, Christos Faloutsos, and King-Ip (David) Lin. Imagemap: an image indexing method based on spatial similarity. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):979–987, 2002.

[123] Harry Plantinga and Charles R. Dyer. Visibility, occlusion and the aspect graph. Technical Report CS-TR-1987-736, University of Wisconsin-Madison, 1987.

[124] Harry Plantinga and Charles R. Dyer. Visibility, occlusion, and the aspect graph. *Int'l. Journal on Computer Vision*, 5(2):137–160, 1990.

[125] J. Ponce and D. J. Kriegman. Computing exact aspect graphs of curved objects: parametric patches. In *Proc. AAAI National Conference on Arificial Intelligence*, July 1990.

[126] J. Ponce and D. J. Kriegman. New progress in prediction and interpretation of line-drawings of curved 3D objects. In *Proceedings of the 5th IEEE International Symposium on Intelligent Control*, pages 220–225, September 1990.

[127] Long Quan and Zhongdan Lan. Linear N-point camera pose determination. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(8):774–780, August 1999.

[128] S. Ravela, B. Draper, J. Lim, and R. Weiss. Adaptive tracking and model registration across distinct aspects. In *Proceedings of the 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 174–180, August 1995.

[129] Aristides A. G. Requicha. Representation for rigid solids: Theory, methods and systems. *ACM Computing Surveys*, 12(4):437–463, 1980.

[130] S. W. Reyner. An analysis of a good algorithm for the subtree problem. *SIAM Journal of Computing*, 6(4):730–732, 1977.

[131] P. L. Rosin. Robust pose estimation. *IEEE Transactions on Systems, Man and Cybernetics*, 29(2):297–303, April 1999.

[132] Marie-Françoise Roy and Thierry Van Effelterre. Aspect graphs of algebraic surfaces. In *Proceedings of the 1993 International Symposium on Symbolic and Algebraic Computation*, pages 135–143, 1993.

[133] S. D. Roy, S. Chaudhury, and S. Banerjee. Aspect graph construction with noisy feature detectors. *IEEE Transactions on Systems, Man and Cybernetics*, 33(2):340–351, April 2003.

[134] M. Sallam, J. Stewman, and K. Bowyer. Computing the visual potential of an articulated assembly of parts. In *Proceedings of the Third International Conference on Computer Vision*, pages 636–643, December 1990.

[135] A. Sanfeliu and K. S. Fu. A distance measure for attributed relational graphs for pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, 13:353–362, 1983.

[136] W. B. Seales and C. R. Dyer. Modeling the rim appearance. In *Proceedings of the 3rd International Conference on Computer Vision*, pages 698–701, December 1990.

[137] Michael Seibert and Allen M. Waxman. Adaptive 3D object recognition from multiple views. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):107–124, February 1992.

[138] M. Sengel, M. Berger, V. Kravtchenko-Berejnoi, and H. Bischof. Fast object recognition and pose determination. In *Proceedings of the 2002 International Conference on Image Processing*, volume 3, pages 349–352, June 2002.

[139] L. Shapiro and R. M. Haralick. A metric for comparing relational descriptions. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 7, pages 90–94, 1985.

[140] Ilan Shimshoni and Jean Ponce. Finite resolution aspect graphs of polyhedral objects. In *Proceedings of IEEE Workshop on Qualitative Vision*, pages 140–150, June 1993.

[141] Ilan Shimshoni and Jean Ponce. Finite-resolution aspect graphs of polyhedral objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):315–327, April 1997.

[142] David Slater and Glenn Healey. The illumination-invariant recognition of 3D objects using local color invariants. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(2):206–210, February 1996.

[143] Humberto Sossa and Radu Horaud. Model indexing: The graph hashing approach. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 811–814, 1992.

[144] Thawach Sripradisvarakul and Ramesh Jain. Generating aspect graphs for curved objects. In *Proceedings of IEEE Workshop on Interpretation of 3D Scenes*, pages 109–115, November 1989.

[145] T. Stahs and F. Wahl. Object recognition and pose estimation with a fast and versatile 3D robot sensor. In *Proceedings of the 11th IAPR International Conference on Computer Vision and Applications*, pages 684–687, 1992.

[146] Peter L. Stanchev and Valery V. Vutov. A model-based technique with a new indexing mechanism for industrial object recognition. In *Proceedings of the 9th IEEE/CHMT International Electronic Manufacturing Symposium*, pages 56–60, 1990.

[147] Louise Stark and Kevin Bowyer. Indexing function-based categories for generic recognition. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 795–797, 1992.

[148] Louise Stark, David Eggert, and Kevin Bowyer. An aspect graph based control strategy for 3D object recognition. In *Proceedings of the 2nd International Conference on Computer Vision*, pages 697–703, 1988.

[149] Louise Stark, David Eggert, and Kevin Bowyer. Aspect graphs and nonlinear optimization in 3D object recognition. In *Proceedings of the 2nd International Conference on Computer Vision*, pages 501–507, 1988.

[150] J. Stewman and K. W. Bowyer. Aspect graphs for planar-face convex objects. In *Proceedings of the IEEE Workshop on Computer Vision*, pages 123–130, Miami, FL, 1987.

[151] John Stewman and Kevin Bowyer. Constructing the perspective projection aspect graph of polyhedral objects. In *Proceedings of the 2nd International Conference on Computer Vision*, pages 494–500, 1988.

[152] John Stewman and Kevin Bowyer. Constructing the perspective projection aspect graph of polyhedra defined using a solid modeler. In *Proceedings of the 6th Scandinavian Conference on Image Analysis*, pages 652–659, 1989.

[153] G. Stockman, G. Lee, and S.-W. Chen. Reconstructing line drawings from wings: the polygonal case. In *Proceedings of the 3rd International Conference on Computer Vision*, pages 526–529, December 1990.

[154] G. C. Stockman, S.-W. Chen, G. Hu, and N. Shrikhande. Sensing and recognition of rigid objects using structured light. *IEEE Control Systems Magazine*, 8(3):14–22, June 1988.

[155] S. Sullivan and J. Ponce. Automatic model construction and pose estimation from photographs using triangular splines. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(10):1091–1097, October 1998.

[156] S. Sullivan and J. Ponce. Automatic model construction, pose estimation, and object recognition from photographs using triangular splines. In *Proceedings of the 6th International Conference on Computer Vision*, pages 510–516, January 1998.

[157] W. C. Thibault and B. F. Naylor. Set operations on polyhedra using binary space partitionaing trees. In *ACM SIGGRAPH 87*, pages 153–162, 1987.

[158] R. B. Tilove. Set membership classification: A unified approach to geometric intersection problems. *IEEE Transactions on Computers*, pages 847–883, October 1980.

[159] F. Toyama, K. Shoji, and J. Miyamichi. Model-based pose estimation using genetic algorithm. In *Proceedings of the 14th International Conference on Pattern Recognition*, pages 198–201, August 1998.

[160] Fatih Ulupinar and Ramakant Nevatia. Perception of 3D surfaces from 2D contours. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(1):3–18, January 1993.

[161] M. A. van Wyk, T. S. Durrani, and B. J. van Wyk. A RKHS interpolator-based graph matching algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):988–995, July 2002.

[162] Aaron S. Wallack and John F. Canny. Efficient indexing techniques for model based sensing. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 259–266, 1994.

[163] C. D. Watkins, S. B. Coy, and M. Finlay. *Photorealism and Ray Tracing in C.* M&T Publishing, 1992.

[164] Nancy A. Watts. Calculating the principal views of a polyhedron. In *Proceedings of the 9th International Conference on Pattern Recognition*, pages 316–322, 1988.

[165] A. Weiss and H. Nawab. A representation for the orientation-dependent appearance of 3D objects. In *Proceedings of the 1988 International Conference on Acoustics, Speech, and Signal Processing*, pages 956–959, April 1988.

[166] Xu Wenli and Zhang Lihua. Pose estimation problem in computer vision. In *Proc. 1993 IEEE Region 10 Conference on Computer, Communication, Control and Power Engineering*, pages 1138–1141, October 1993.

[167] R. C. Wilson, A. N. Evans, and E. R. Hancock. Relational matching by discrete relaxation. *Image and Vision Computing*, (13):411–421, 1995.

[168] R. C. Wilson and E. R. Hancock. Structural matching by discrete relaxation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(6):634–648, June 1997.

[169] Andrew K. C. Wong, Si W. Lu, and Marc Rioux. Recognition and shape synthesis of 3D objects based on attributed hypergraphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(3):279–290, March 1989.

[170] K. C. Wong, Y. Cheng, and J. Kittler. Recognition of polyhedral objects using triangle pair features. In *IEEE Proceedings on Communications, Speech and Vision*, volume 140, pages 72–85, February 1993.

[171] P. L. Worthington and E. R. Hancock. Object recognition using shape-from-shading. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(5):535–542, May 2001.

[172] M. W. Wright and F. Fallside. Object pose estimation by neural network. In *Porceeding of the 1992 International Conference on Image Processing and its Applications*, pages 602–603, April 1992.

[173] R. Yagel, D. Cohen, and A. Kaufman. Discrete ray tracing. *IEEE Computer Graphics and Applications*, 12(5):19–28, September 1992.

[174] Youngrock Yoon, G. N. DeSouza, and A. C. Kak. Real-time tracking and pose estimation for industrial objects using geometric features. In *Proceedings of the 2003 IEEE International Conference on Robotics and Automation*, pages 3473–3478, September 2003.

[175] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

[176] H. Zha, T. Shibata, and T. Nagata. Recognition of polyhedral objects – aspect graph generation based on a learning-by-showing approach. In *Proc. IEEE International Conference on Systems, Man, and Cybernetics*, pages 713–718, October 1996.

Index