OBJECT–ORIENTED PROGRAMMING FOR IMAGE ANALYSIS

Dietrich W. R. Paulus and Heinrich Niemann

Universität Erlangen–Nürnberg

Lehrstuhl für Mustererkennung (Informatik 5)

Martensstraße 3, D–91058 Erlangen, Germany

email: {paulus,niemann}@informatik.uni-erlangen.de

This is a reformatted version of the preprint as cited in the self–reference [47]. The format of the bibilography has also been changed.

OBJECT–ORIENTED PROGRAMMING FOR IMAGE ANALYSIS

Dietrich W. R. Paulus and Heinrich Niemann
Universität Erlangen–Nürnberg
Lehrstuhl für Mustererkennung (Informatik 5)
Martensstraße 3, D–91058 Erlangen, Germany
email: {paulus,niemann}@informatik.uni-erlangen.de

Abstract

In this survey we discuss the implications of object–oriented programming on image processing, image analysis, and real time active vision. We give an overview on the important literature which relates to object–oriented programming and imaging.

A general object–oriented framework covering all levels from image capturing over segmentation, model generation up to object recognition is presented and related to other well known systems like Khoros, KB Vision, IUE, and PIKS. In particular, we describe real time features and 3D processing. We illustrate the main ideas of this system by solutions to real world image processing problems.

1 Introduction

The programming language C++ [63] had an overwhelming success in the last few years in all areas of computer science. Although the C++–language is only partially suited for object–oriented programming and although it has its disadvantages [60], it presently seems to be the best choice to combine efficiency with object–oriented design in real world applications, especially those operating under real time conditions. Efficiency and maintainability are of course crucial for most image analysis programs.

A key mechanism of object–oriented programming is *polymorphism*. In this article we give examples of how polymorphism can simplify image processing programs. After a short introduction of the general terminology of object–oriented programming in Sect. 2, we describe the architecture of image analysis systems (Sect. 3) and show the effects on image processing and analysis (Sect. 4).

Early image processing systems were mostly written in Fortran (e.g. SPIDER [66]). One decade was dominated by the use of C (e.g. [15]). The discussion of object–oriented programming for image processing started when C++ became known. This programming language promised to provide the efficiency required for image processing, together with object–oriented programming, and the possible code reuse by the upward compatibility to C. This leads to various joint efforts for a common image understanding environment. We discuss the present state of this research in Sect. 4 and relate it to other systems and standards.

In general, image analysis has to use knowledge about the task domain. This knowledge is represented in semantic networks in a straight forward way. We describe an object–oriented implementation of a knowledge based image analysis system in Sect. 5.

Several examples and applications in Sect. 6 prove the feasibility, efficiency, and reusability of object–oriented programs in image analysis. In

Sect. 7 we summarize the resource requirements of the applications.

2 Object–Orientation

In this section we briefly introduce the important ideas and terms of object–oriented software development.

2.1 Programming Languages

Object–oriented programming has been known to computer scientists since 25 years. The ideas originated in the ancestors Simula [3] and Smalltalk [24]. During this period of time, the programming language C had its breakthrough in the world.

In the late eighties, the C language was extended with object–oriented ideas. The language Objective–C [11] encapsulated Smalltalk features in the C language. The language C++ [63] mainly used the ideas of Simula; a public domain class hierarchy called NIHCL [25] incorporates some Smalltalk ideas into C++. C++ and Objective–C are supersets of C; i.e. every C program is a C++/Objective–C program as well. Many valuable image processing routines nowadays written in C can be reused without modification. Possibly because of the cheap or free availability of C++–compilers — even on personal computers — this language had enormous success.

Reuse of class libraries like NIHCL serves for two purposes. Common programming problems — like the implementation of linked lists or sets — have already been solved in these systems and can be used without further effort. Exchange of software using such class libraries with other institutes is simplified since the classes share the same structure and interfaces.

2.2 Advantages

Object–oriented programming is used to reduce difficulties of conventional software development. In particular, software reuse is highly desired due to the high costs of programming. Modularity is a central concept which helps maintaining large systems. Data abstraction provides clean interfaces which are essential when several programmers share code in a team.

These goals can be reached in conventional programming with certain effort. In object–oriented programming, they are intrinsic to the programming paradigm; it would require certain effort *not* to fulfill these principles.

One goal of software engineering is to provide components with a long lifetime, even when changes are required. Object–oriented design can not guarantee this, but it simplifies updates and evolution.

The programming language C++ in particular has the advantage that it combines efficient conventional constructs with object–oriented features. Existing routines in C which sacrifice clean structure to gain speed — which unfortunately is necessary in some rare cases — can be encapsulated in classes or objects which provide safe interfaces.

Software reuse is crucial in large systems; well documented packages should be usable even across applications. If every programmer is allowed to re–program existing code, soon several pieces of code will be scattered around in the system which serve the same purpose — with minimal differences which soon nobody will remember exactly. The system becomes hard to maintain, then.

OOP (object–oriented programming)simplifies code reuse by inheritance and documentation by the hierarchical structure of classes. In addition to the technical advantage of software reuse, it may as well be of scientific interest (Sect. 6.2).

Generally, methods in object–oriented programming have fewer arguments than corresponding function calls in traditional programming, since parts of the required information may be already bound to the object (for example, a FFT object may have its internal tables for the the acutal image size, and no such tables will have to be allocated for the function call and passed to it as arguments). This again facilitates software

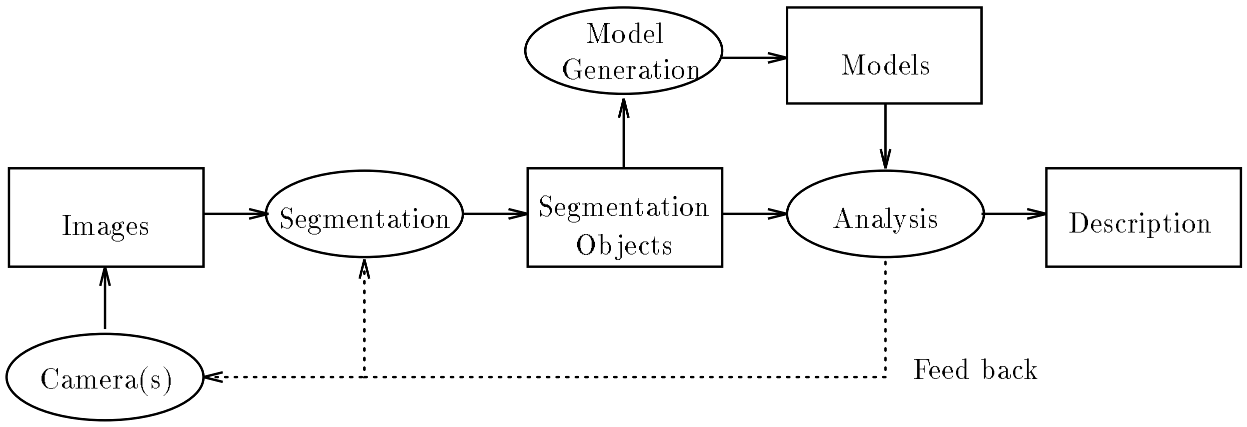

Fig. 1: Data flow in an image analysis system

maintenance and reuse, especially if a class and its interface has to be exchanged. Fewer modifications are then required in the code, compared to conventional programs.

3 Image Understanding

In this section we describe the general software architecture of image understanding (IU) systems and apply object–oriented principles.

3.1 Data Flow

The problem of image analysis is to find the best description of the input image data which is appropriate to the current problem. Sometimes this means that the most precise description has to be found, in other cases a less exact result which can computed quicker will be sufficient. This task may be divided into several sub–problems. After an initial preprocessing stage, images are usually segmented into meaningful parts. Various segmentation algorithms create so called segmentation objects [52].

Segmentation objects are matched against models in a knowledge base which contains expectations of the possible scenes in the problem domain; this knowledge may be generated automatically from samples in a training phase [62, 70].

The various data types involved in image segmentation, like images, lines, regions, etc., may serve for data abstraction of the problem. In object–oriented programming, these data types are naturally represented in classes. Segmentation may be seen as a data flow problem relating these various representations. An overview of the main components is shown in Fig. 1; data is captured and digitized from a camera and transformed to a description. Image processing tasks are shown in oval boxes; data is depicted as rectangles; in the object–oriented design phase, both will naturally be grouped to class hierarchies.

The problem of finding an optimal match and the best segmentation can be seen as an optimization problem and is formulated as such in [43]. Optimization may search for the best possible match [71] as well as include efficiency considerations which are crucial in real time image processing. This process may require a data feed back from high–level processing to data driven segmentation, or even to the image capturing devices (dotted line in Fig. 1). This data flow is also common to active vision systems, where the optical parameters are adjusted by the controlling computer in order to get the best possible image for the actual analysis purpose [14].

3.2 General Structure

Knowledge based image analysis extends the data flow shown in Fig. 1. The general modular struc-

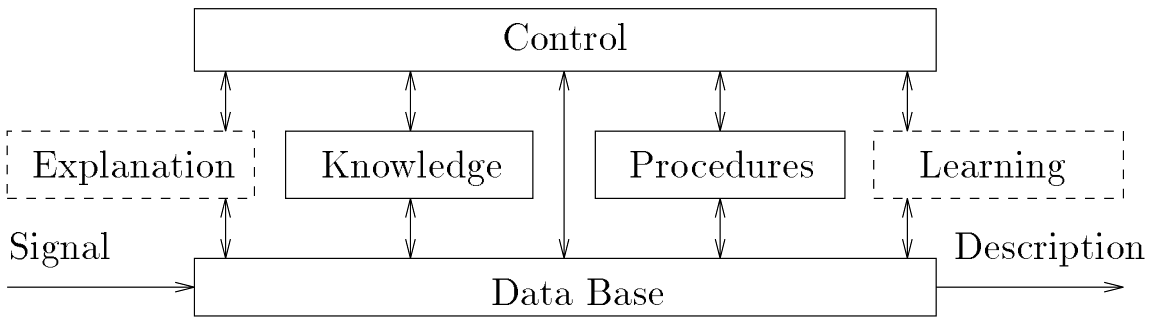

Fig. 2: General structure of image analysis systems

ture is shown in Fig. 2; no separation between tasks and data has been made here. Each module has separate responsibilities. The knowledge base is contained in a separate module. A module for procedures provides all required operations for management of the knowledge base and for image analysis. A general control module selects procedures based on the information in the knowledge base and on the actual image data. Learning and explanation are useful but not required for some applications.

Knowledge about the problem domain is mostly represented as models for physical objects, tasks, cameras, lighting etc. Object models are matched against segmentation data (Fig. 1). The search algorithm (e. g. the A^* algorithm) for solving this optimization problem is independent of the image processing algorithms (module procedures); it is part of the control module. Model generation (Fig. 1) is the task of the learning module. The explanation module may provide textual descriptions of the actions carried out as well as visualize the various stages of processing.

A model of the scene is used as knowledge for image analysis. The scene may be decomposed into object–models which can be represented using their structural properties and relations (e.g. in a semantic network), or as statistical object models ([30], Sect. 6.2). These models must be matched against segmentation data during analysis.

The system architectures in Fig. 1 and Fig. 2 imply various interactions between modules. Information has to be exchanged by means of well defined interfaces. Modules and data structures with well defined interfaces are naturally implemented as classes in OOP.

The segmentation object is a central idea for the data representation independent of the algorithms used for image segmentation, and can be used as an interface between data–driven segmentation and knowledge–based analysis. Models can be described in a similar formalism [42].

3.3 Real Time

Image *sequences* were studied for motion detection. One actual goal of image analysis is to process images as they are delivered from the camera, i. e. in video rate and real time. After a fixed period of time, all iconic information will be lost due to the limited memory capability of machines (as well as humans); only symbolic information may be kept longer.

In OOA (object–oriented analysis),the problem will be decomposed; one component will be the image sequence. In OOD (object–oriented design),general sequence classes and image classes will be combined to new classes. In OOP, the internal management of a ring buffer will be implemented which holds the images in memory.

Real time processes typically have to work for a long period of time. Computing resources, e.g. storage in the computer, are limited. Thus, real time processes have to guarantee constant resource demands. In addition, quick and reliable response times have to be confirmed by the software. These demands are imposed on the individual components of the software. For example, the image ring buffer will have to specify its maximum switching delay time. Operators on images have to release all resources which are no longer needed or referenced. If symbolic information is to be kept longer, safe interfaces have to exist for a partial or full release of this information, i.e. there has to be an explicit means for forgetting symbolic information.

3.4 Devices and Actors

The output of the system in Fig. 2 is a description of the input image data. In active vision systems, the output may additionally or alternatively contain control commands for the sensor device (e.g. a change in the aperture of the lens) or for the actor (e.g. a robot or a moving vehicle). The system design in Fig. 2 is therefore suitable for conventional image analysis as well as for active vision.

Interaction with graphical and pointing devices is necessary for the explanation module. Interaction with physically moving tools requires a control loop which can be closed in real time.

Naturally, these actions and devices are modelled as classes and objects in OOA and OOD. Their well defined interface facilitates data exchange between programs and devices.

3.5 External Representation

Data exchange between different processes often uses the facilities provided by the operating systems, i. e. shared memory, files, and pipes. When such processes run on different hardware, the file format has to be machine–independent. In an object–oriented environment this is strongly connected to the problem of persistent objects, i.e. objects which remain after termination of a process.

Normally, image files are fairly big; binary storage is mostly required when they have to be saved on permanent media or have to be transmitted on slow communication channels. Several machine independent image file formats have been proposed [5, 4, 8] or established, like `tiff` [55]. XDR [64] provides machine independent binary storage for arbitrary data types.

The JPEG [67] and MPEG [20] data encoding schemes presently play an important role in the areas of graphics and multi media processing. They are less important for image analysis, since this kind of data compression discards high frequency information which is regarded as essential for image segmentation. Only lossless data compression schemes can thus be used for image storage. Such compression is available for example in `tiff`.

4 Systems

We now discuss the influences of object–oriented programming to a selection of existing image processing and analysis systems.

4.1 Display and Edit

Several Unix applications exist for image display, manipulation, conversion, etc. Most of them are in the public domain or share ware. To name some of them: xv, image magic, animate, pbmplus, etc.[1]

Graphics editors can be used to create images from high–level descriptions. The program Xfig is commonly used in Unix. These programs are written in C and use X11. Object–oriented applications can also be found. The Interviews system [40] is written in C++ and provides display and editors.

Image editors for manipulation of sub–images are presently missing. They are however available as commercial PC products.

These tools can read, convert, and write almost any image format on disk.

[1]xv by John Bradley, Image Magic and animate, by John Cristy, the portable bitmap utilities (pbm) by Jef Poskanzer; check your nearest ftp site for the latest sources.

4.2 IKS and PIKS

The idea of a common tool box for image processing came up when image processing was established as a discipline with a set of basic routines. SPIDER [66] was a commercial step in this direction. Soon after graphics was standardized in the "graphical kernel system" GKS system [17], people started working on an image processing counterpart, the "iconic kernel system" IKS [39] which later lead to the PIKS "programmer's imaging kernel system" [6, 8].

Several reasons beyond the scope of this article delayed the launch of such a system. Several institutes collaborated in this idea [5, 9, 21, 56]. During this development, object–oriented ideas were discussed [7, 23], but they were mostly discarded in the final standard by DIN, ANSI, and ISO [36]. The PIKS system provides a common imaging model and an application programmers interface for image processing. A set of several hundreds of operators is defined which operate on multi–dimensional multi-band images. All functions which are commonly found in commercial systems are included. An appropriate encapsulation in classes will be required for object–oriented programming. Since this is not part of the standard, it will add new diversity to the software interchanges, contradicting the efforts of a standard.

Along with the standard came a new image interchange format [4, 8]. Using this specification, images can be exchanged across application boarders, like from printing technology to robotics, and vice versa. Because of the overhead required by the generality of the format, the format is suited for data exchange, not for local short term storage.

4.3 HORUS

The HORUS system [16] is an image understanding environment with interactive graphics running on various platforms. Interactive image manipulation in non–rectangular areas of interest is possible. A basic C++ interface exists; the lower level of the system is implemented in C. Higher–levels can use Prolog, Lisp, or Smalltalk. Object–oriented programming is available via the Smalltalk interface which encapsulates underlying C–functions.

The library for image processing consists of approximately 460 operators for 2D image processing covering all areas from preprocessing, segmentation, morphology, feature extraction, classification up to 2D model generation. A knowledge base for image analysis is provided and represented as a semantic network.

4.4 Khoros

The Khoros system [58] is an environment for interactive development of image processing algorithms. The system includes a neat visual programming environment. The algorithms can also be run without interactive graphics. The system provides a large library of imaging functions (over 500 functions), some of them used in $2\frac{1}{2}$D and 3D image processing. Knowledge based processing is not part of this package.

Khoros is suited for Unix workstations. Mean size of the 291 binaries in the distribution (Release 1.0) is 0.55 megabyte under HP–UX 9.01 using shared X11 libraries. Khoros is in the public domain and is presently written in C. Version 2 has ANSI C header files and can be used with C++, although it is still written in C.

Khoros is working on real world applications. It does not contain hardware dependent interfaces like cameras or display devices.

The Khoros algorithms are very general and mostly operate on several spectral channels as well as on monochrome images. This generality leads to a considerable run–time overhead making real time performance almost impossible.

4.5 KB Vision

The KB Vision (KBV) system [68] is a commercial product loosely based on the VISIONS system [26, 27]. It covers all areas of knowledge based image analysis from signal processing up to image understanding. VISIONS uses semantic networks for knowledge representation [59]. Initial symbolic information is accessible to the user in KBV as a "token set" for which powerful management tools exist.

KBV includes a windowed environment for all kinds of display and interaction. Many functions are provided in the library. As with Khoros, the generality of processing decreases speed. In particular, access of a single pixel is very slow compared to pointer operations in C.

4.6 IU Environment

A group of leading experts in image analysis joint their efforts for a common image understanding environment [41, 28]. A system is planned as a common basis for image processing and knowledge based analysis. Applications may be written in C++ or LISP. The system covers all areas of imaging with many applications; due to the many contributors, a variety of ideas has to be united into a common hierarchy of classes. The design goals are: object–oriented programming with graphical interaction, extensibility and program exchange and a common performance evaluation platform for image processing. Real time processing is explicitly excluded from these goals. This is reasonable because the generality of the class hierarchy might slow down processing.

In the present draft, no classes are provided for devices such as cameras. The C++ part which needs a general object–oriented environment (Sect. 2.1) presently included classes for sets, stacks, lists, etc., i. e. existing systems for this purpose (e. g. NIHCL [25]) are not reused. The system is currently under construction. The first estimate of over 100 classes [41] will probably be exceeded considerably.

4.7 View Station

A software architecture planned as a common basis for program exchange between companies, research institutes, and universities is presented in [37, 61]. This system extends the ideas of SPIDER [66] and offers more than 500 algorithms for image processing. It includes a visual programming environment, windowed graphics and a new image storage scheme. Special memory–mapped image hardware can be encapsulated. Imaging devices are thus accessible by the operating system interface. Object–oriented programming is used here to gain a hardware independent software structure.

The system was developed by a company which manufactures work stations. However, the aim of this system is to provide a *portable* platform on contemporary Unix workstations with transparent access to fast dedicated image processing hardware. For the company's machine, special hardware acceleration is encapsulated in classes.

The system is written in C++ and requires a Unix workstation with graphics display. It includes an interpreted high–level language for image analysis. A programming interface allows access, use, and extension of existing C++–classes. Real time processing is possible using the dedicated hardware.

4.8 Animals

A fully object–oriented approach was chosen for the Animals system [46], which is built along the data flow in Fig. 1. We describe the system in Sect. 5. Image processing data is represented in a hierarchy of picture processing objects which is presented in Sect. 5.1. Algorithms are also represented as classes and structured hierarchically (Sect. 5.2). A knowledge base uses the semantic net formalism and is implemented as a hierarchy of classes as well (Sect. 5.3).

Devices for image processing, such as cameras or displays, are encapsulated in classes (cmp. [7, 18, 23]). No fancy graphical environment exists. The system is oriented towards efficient unsupervised batch processing and active vision. Image classes may be compiled with safe access mechanisms; for real time applications they can exhibit access mechanism identical to two dimensional dynamic arrays in C, alternatively. The system also includes a robotic part combined with a closed sensor–actor loop. We present applications in Sect. 6. Visualization of images is done with X11 tools, the results of segmentation can be converted to various graphics formats for display purposes (cmp. Sect. 4.1).

4.9 Comparison

The different features of the systems introduced in the previous sections are summarized in Tab. 1. Reuse of existing code is documented for Khoros (line "reuse" in Tab. 1) which uses LINPACK functions. The View station extends the SPIDER library which was used by many programs and in this sense helps code reuse. Animals uses the "Numerical Recipes" [57].

	HORUS	Khoros	KBV	IUE	View station	Animals
classes	no	no	no	yes	yes	yes
reuse	$*$	yes	$*$	$*$	yes	yes
graphics	yes	yes	yes	yes	yes	no
real time	yes	no	no	no	yes	yes
knowledge	yes	no	yes	yes	yes	yes
devices	$*$	no	no	no	yes	yes
language	C	C	C	C++	C++	C++
format	$*$	viff	$*$	$*$	I/F	xof
price	> 0	0	> 0	0	0	0

Tab. 1: Summary of features ($*$ means unknown)

Neither IUE nor the View station build their class hierarchies on top of an existing object–oriented system for C++, i. e., they have to provide their proprietary tools for object–oriented programming in C++, like sets, collections, lists, etc. Animals reuses NIHCL for this purpose (Sect. 2.1).

Khoros is in the public domain; a news group is used as a bulletin board for problems and solutions. The KBV and HORUS systems are sold commercially. The support of the IUE will have to be discussed in the future; it is planned to put contributions to it into the public domain. The IKS resp. its successors are documented as (international) standards; changes or updates have to follow the regulations for this subject. The View Station system is free software. The Animals system is available *as is*, i. e. without service.

The other rows of Tab. 1 will be mentioned in the following sections.

5 An Image Analysis System

We now describe **An im**age **a**na**l**ysis **s**ystem (Animals, Sect.4.8) in more detail. The system is written in C++; it is integrated into a larger system for pattern analysis (signals, speech, as well as images) called PUMA ("**P**rogrammier **U**mgebung für die **M**uster**a**nalyse", programming environment for image analysis).

5.1 Data Representation

Various data representation schemes have been developed for data in image processing. Some of them may be treated as algebras with more or less complete sets of operations (e. g. chain codes or quad trees). Other representations are used because of their storage efficiency (e. g. run length codes), others because of their runtime efficiency. Such ideas were combined into a **Hi**erarchy of **P**icture **P**rocessing **O**bject**S** (HIPPOS, written as ἵππος [46]).

A central problem visible in Fig. 1 and 2 is the data exchange of segmentation results, which are an initial symbolic description of the image. The solution in ἵππος is a hierarchy of classes, which is shown in Fig. 3 in the graphical notation of

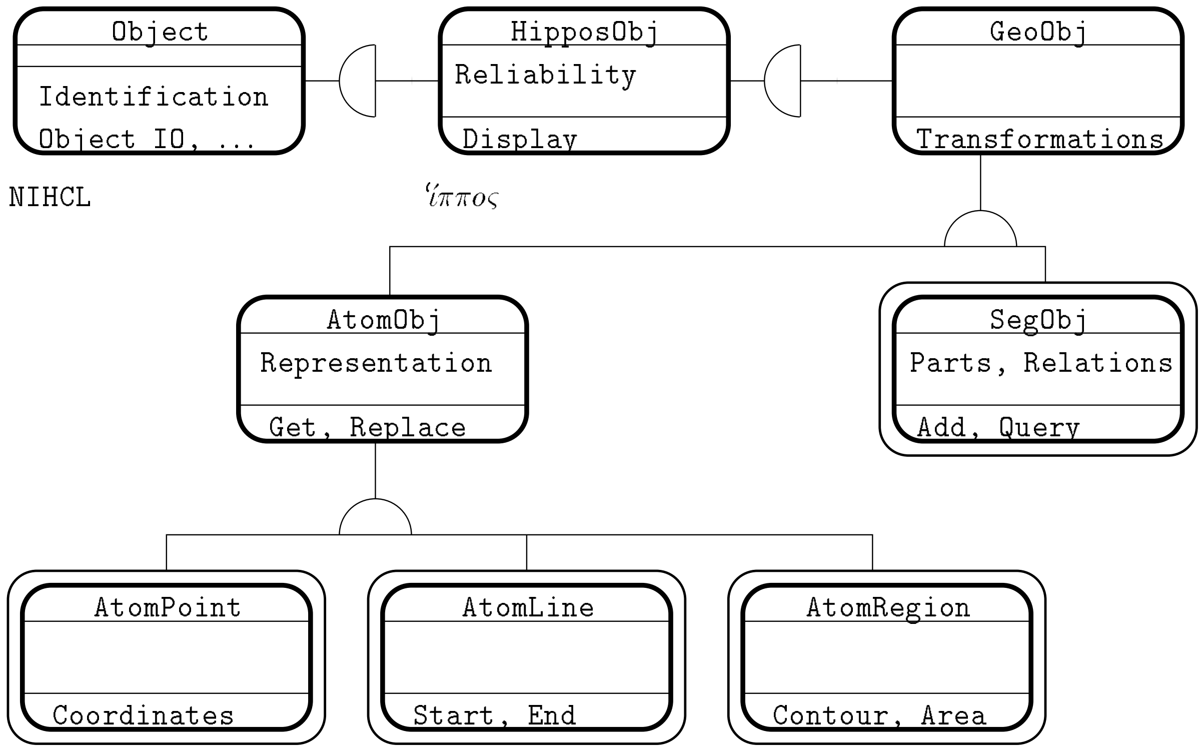

Fig. 3: A section of a hierarchy of geometric objects for segmentation (in total approximately 100 classes)

[10]: class names are the top entries in the boxes; the center field contains important attributes, and the lower field lists a selection of relevant methods; both may be empty; with dots we mark that there exist methods or attributes which are not mentioned in the text. Specialization (inheritance) links are marked with a semi–circle round towards the general class; single boxes mark abstract classes, double ovals are concrete classes.

All segmentation results can be stored in an object of class `SegObj` [50, 46, 52], no matter whether the data is computed from line–based or region–based segmentation, color–, range–, or gray level images, etc. This general tool is used in many algorithms. Several subtrees group representations for points, lines, regions, surfaces, and volumes. A segmentation object is a geometric object (`GeoObj`). It consists of a set of parts which are in turn geometric objects. Atomic objects are geometric objects as well and end the recursion. The abstract base class `HipposObj` connects to the NIHCL object and serves as a common root for all image processing classes. It contains a judgement attribute which is used for the comparison of competing segmentation results by the analysis control (Sect. 3.2).

Every atomic object has an associated representation, such as a chain code for an atomic line. These representations are organized in a separate hierarchy (Fig. 4). Classes for 2D are separated from surface representations and 3D objects. If 2D objects were considered special cases of the corresponding 3D case, e.g. if a 2D point were a 3D point with $z \equiv 0$, all 2D objects would carry the overhead for storage and 3D computation with them. On the other hand, if 3D objects were derived from 2D objects, this would contradict the intuitive meaning of inheritance — which is specialization. Having objects with arbitrary dimension would as well slow down computation and increase storage requirements. The same arguments hold for the time dimension. Details can

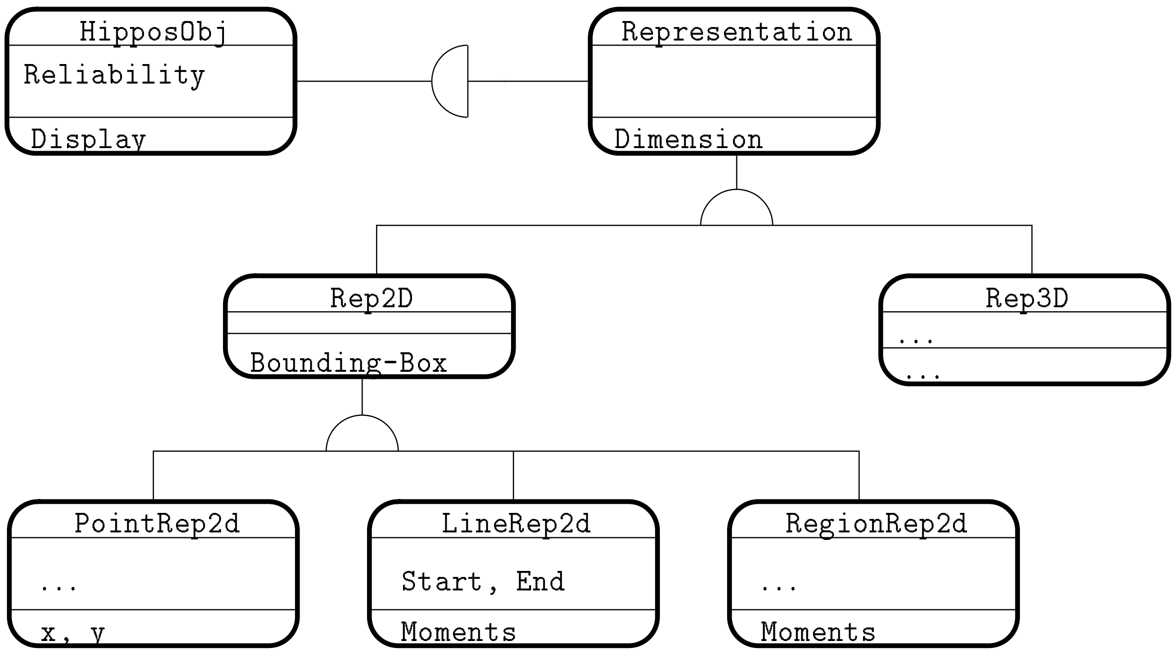

Fig. 4: Hierarchy of geometric objects for segmentation

be found in [46].

The representation tree is fairly large to hold most commonly used data structures for image segmentation, such as chain codes, polygons, or quad trees; only some abstract base classes which bundle common interfaces are shown in Fig. 4. They give an example, the derivation scheme for the chain code class is `Chain` → `BasicLineRep` → `LineRep2d`. Since atomic objects mostly delegate their actions to the associated representation objects, the methods in Fig. 3 and Fig. 4 are similar. Segmentation objects can be used in real time image processing; methods are provided for releasing parts or all of the information, as described in Sect. 3.3.

As the applications (Sect. 6) show, this has proven adequate for 2D, $2\frac{1}{2}$D and 3D image analysis. Another sub–tree exists for image classes, like gray level images, stereo images, range images, color images, etc. The whole *ἵππος*–hierarchy currently consists of approximately 100 classes. Additionally, classes for homogeneous coordinates are used for matching in knowledge based analysis.

5.2 Operator Hierarchy

Many operations in image processing can be structured hierarchically in a straight forward manner. Some edge detection operators can be specialized to edge mask operators; one of them is the Sobel operator. Filters can be linear or non–linear; rank order operations are one type of non–linear operators; the median is one special case of this type.

This hierarchy can be implemented in a hierarchy of classes for *operations* in a straight forward way (cmp. [7, 18, 23]). Objects are the actual algorithms with specific parameter sets [29] which are also objects. This is particularly useful, when operations have internal tables that increase their efficiency. As an example consider a discrete Fourier transform which uses tables for sine and cosine values; the size and contents of these tables varies however upon the frame size to be transformed. Several Fourier–transform objects may thus be used in one program for transformations of images, e. g. in a resolution hierarchy. In contrast to conventional solutions, this requires neither code duplication nor complicated management of functions with local storage.

Fig. 5 shows a simple hierarchy for filter operators. The characteristics may be high–, band–, or low–pass and can be enquired by the method for operator description. A linear filter can be described by its convolution kernel. No specific information is provided in the abstract class for non–linear filters which is only used to group the derived classes, such as a k–nearest neighbor filter or a rank order operation.

Dynamic enquiry about available operators in the system is also facilitated by the operator hierarchy. Parameter blocks can be shared between different operators as parameter objects. Function call C++ syntax for these operators is used in Animals which facilitates migration of existing conventional programs to operator objects.

The sequence of operators introduces a refinement of the data flow in Fig. 1. Intensity or range images are transformed to feature domain images, sets of lines, corners, vertices, and finally to segmentation objects [52].

The major advantages of operator–classes are twofold. Algorithms can be programmed in an abstract level referencing only the general class of operations to be performed; extensions of the system by a new derived special operator will not require changes in the abstract algorithm. Dynamic information about the operator, which is actually used, is available. For example, a program may just reference a filter object; during run time it will be decided which concrete filter should be used. Using virtual functions in C++, the run time overhead is negligible.

When the message "apply" is sent to an operator object together with image objects as arguments, new images are created as a result. Other operator objects exist for segmentation. The transitions in Fig. 1 can thus be programmed using image objects, segmentation objects, and operator objects.

For experiments, the operators (Sect. 5.2) are encapsulated in processes which exchange persistent objects. Persistent ἵππος–objects (in fact,

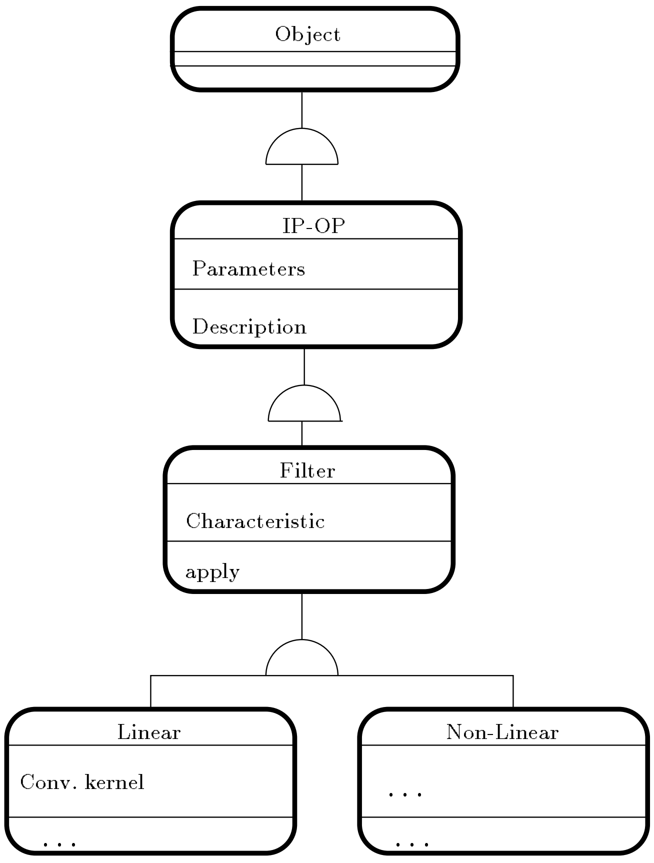

Fig. 5: Operator hierarchy for image processing

also the NIHCL objects) are created using methods for an XDR representation (Sect. 3.5) which can also be used for process communication with remote procedure calls (RPC). We call this format an external object format (xof, Tab. 1). Too many image formats already exist; without need no new scheme should be defined. In our case, the combination of object–oriented programming and image processing clearly required such a definition. Images and other objects — like lines, regions, segmentation objects — are represented using the same format. This again — as in the case of operator classes — facilitates programming and code reuse. For example, a display module can read an arbitrary object from a file and display it with the object's method, no matter whether it is a line or an image; new objects will be known without change in the program source. The same ideas lead to a proprietary format for the view station (Sect. 4.7).

For applications, the operators can be linked together to one process; then, no external communication is required.

5.3 Knowledge Based Analysis

It is commonly agreed that image analysis has to rely on knowledge about the particular task domain. Semantic nets are a general tool for knowledge representation used in several image analysis systems, e. g. [16, 44, 59]. A semantic net formally is a labelled graph; nodes are concepts, edges relate concepts to each other. The ERNEST system [44] is a knowledge based system using semantic nets which was applied in several image analysis applications. Six types of edges are used in ERNEST: concrete, specialization, part, referential, model, instance. Details can be found in [44].

An object–oriented implementation of semantic nets using the ideas of ERNEST has been realized [71, 74]. The control algorithm (Fig. 2) may now be either sequential or parallel [19]. The basic idea is to provide abstract super–classes for concepts, instances, and modified concepts (concepts constrained by the current state of the analysis). User defined concepts are translated to classes from a formal language [48], which are derived from those abstract classes. Analysis states are defined as classes to provide an abstract interface for the control module. Analysis can be defined as the search for an optimal state sequence. The control relies on judgement classes which associate a distance measure from the goal with each state during the analysis process.

5.4 Model Generation

Computer models for objects are also used in industrial production. CAD descriptions of intuitively simple parts may still have complicated structure. It may be a time consuming task to create such models.

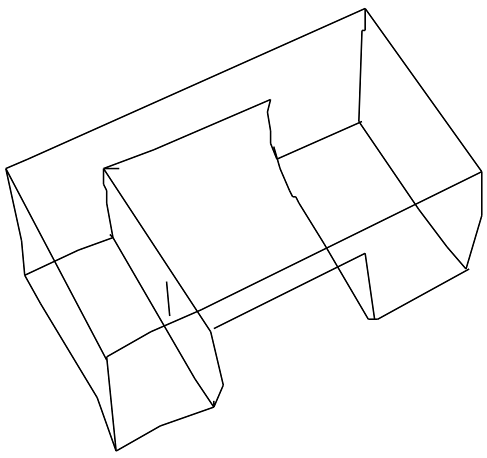
Fig. 6: Intermediate 3D line model of a polyhedral object

Models for knowledge based image analysis share the same problem. The goal thus has to be to create the models automatically. This is the task for the model generation step (Fig. 1) which results in a hierarchy of classes for models [49, 72]. The result of an automatic generation of a polyhedral model is shown in Fig. 6.

Imagine a system for the recognition of simple 3D geometric objects. Models for all kinds of such objects will be needed, like a cube, a pyramid, a cylinder, etc. [73] These descriptions can use surfaces, vertices, lines, and relations. This common structure is described in the model scheme (–classes) which has to be provided by the user in the formal language (Sect. 5.3) describing a semantic net. The actual model is generated from a sample of images, where the relative positional transformation of the camera from one image to the next is roughly known [49, 72]. This can be accomplished by an active change of the camera position mounted on a robot [12]. The line–model in Fig. 6 is computed from 3D lines [69, 73]; stereo images [38, 51] or range images are segmented for this purpose [29], both represented with the same programming tools as segmentation objects. The 3D lines are integrated into one model; a final model for polyhedral objects can be computed, if planar surfaces are approximated. An extension to polygons is described in [72].

Since models are generated from segmentation data, automatic model generation insures that models consist of descriptions which can be effectively computed and matched against segmented images. This approach shows how structural properties of an object can be learned and represented explicitly. Another approach is to model these properties statistically ([30], Sect. 6.2). Various modules of the object–oriented system interact and exchange information for model generation. To mention some of them camera classes, semantic network classes, and segmentation objects are used.

5.5 Active Vision

New constraints are imposed on the software structure of image analysis systems by the new active vision paradigm which is summarized in [65]. Since many active vision techniques require a feed back from analysis to low level processing (Fig. 1), they have to operate in real time. Changes in camera parameter settings — like a change in the aperture of the lens or focus — are a common active vision technique. Also, positional changes of the optical system are desired which are accomplished if the camera is mounted on a robot [12].

The implications for an object–oriented software structure are straight forward. Since real time processing is required, objects have to exhibit clearly defined requests for resources, like memory, time, or devices. Objects have to release all resources which are not used elsewhere upon object destruction. The principles and tools of object–oriented programming facilitate it to obey to these strategic regulations. Devices such as cameras or robots are needed as classes. This provides mostly hardware independent access. Machine independent software, of course, increases software reusability which is crucial for large software systems on rapidly changing hardware platforms. If such a device is to be replaced, the software interface should remain unchanged; this is simplified by object–oriented programming (Sect. 2.2).

The separation of algorithmic code for image processing and analysis from the code which interfaces to the operating system and to the dedicated devices helps maintenance of the programs by the modular structure.

6 Applications

We now demonstrate the flexibility of Animals (Sect. 5). Experiments for object recognition, tracking, and image analysis are also shown. In traditional programming, new problems are solved with a combination of new and old functions. In Animals, new classes and functions are combined with existing ones.

6.1 Object Tracking

An experiment carried out under real time constraints is described in [12, 13, 14]. A moving toy train is to be followed by a camera mounted on a robot (overview in Fig. 7). Active vision techniques are applied to reach this goal on two standard Unix workstations in parallel.

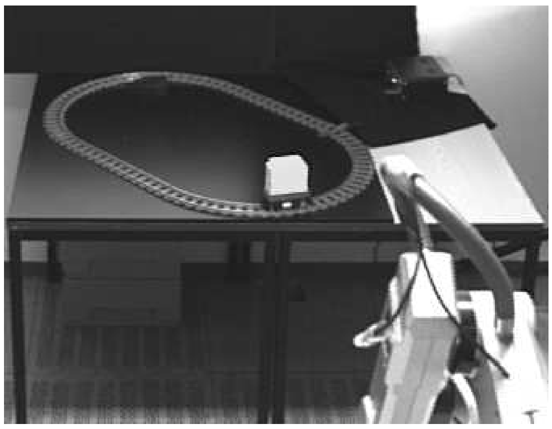

Fig. 7: Robot (one arm in the front) and moving object (toy train in the back)

A two stage object tracking module operates in a closed loop of robot control and image processing. In the first stage, the moving object is detected in an overview image. The second stage tracks the object in a region of interest which is generated by an active change of the camera device object. The system changes the camera position continuously, such that the moving object is always in the center of the image. The interface to the robot is available as a robot–object.

Selective processing on sub–image (–objects) speeds up the computation. Snakes are used for tracking and implemented as classes. Fig. 8 shows a snake computed in the down–sampled sub–image captured from the robot's camera. Current research incorporates object features such as edges and straight lines into the computation of the snake, resp. the minimization of the energy function for the snake.

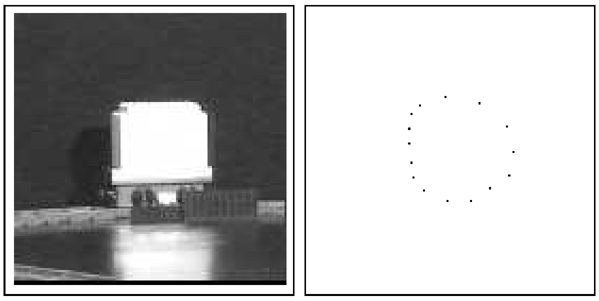
Fig. 8: Snake around the moving object

This application shows that real time constraints can coexist with general object–oriented software tools and a modular system in which separate tasks are distributed to separate processes. Currently, the speed of the system is limited by the frame grabbing hardware, not by the image processing software.

Active changes of the camera device are programmed on an abstract level. The same interface is used for several different cameras and input lines.

6.2 Statistical Object Recognition

Statistical methods for object recognition are presented in [30, 32, 31]. In [32] we show the advantages and limitations of Hidden Markov Models (HMM) for 2D image analysis. The limitations are mainly due to the prerequisite of HMM's which requires a linear sequence of input features. This order has to be artificially superimposed to affine invariant form features. A class hierarchy for HMM's was integrated into PUMA (speech as well as image processing). The well known algorithms for HMM's like the Baum–Welch or Viterbi–algorithm can be programmed in the base class and are available for all special kinds of HMM's. Again, code reuse is simplified. On the other hand, existing matrix classes had to be extended to include the mathematical operations needed during the training phase. These operations are now available to all users of PUMA. The HMM classes can be used in speech applications as well as image analysis. Code reuse is provided for different groups of programmers. Remarkably, this goes hand in hand with personal communication and exchange of scientific ideas, not only program code!

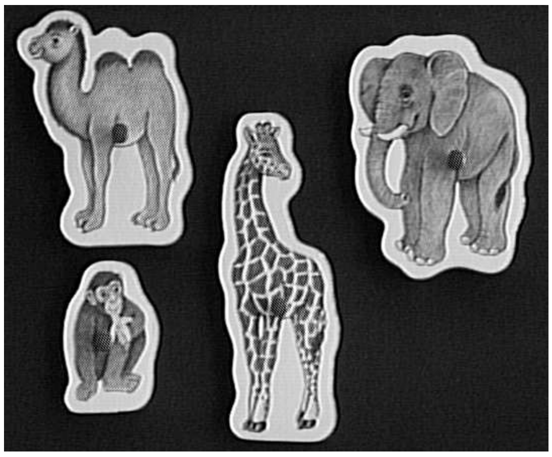
Fig. 9: Input image (some animals (!))

An input image for this image processing application is shown in Fig. 9. Segmentation results consisting of polygons which are used for the detection of form features, are shown in Fig. 10.

In [31], the mathematical details of 3D statistical object recognition are explained; first results are presented in [30]. In this approach, statistical object models are represented as mixture density

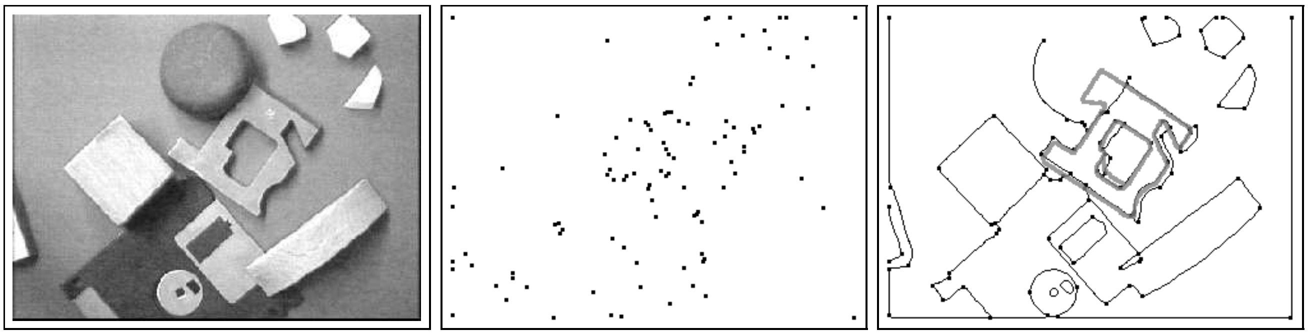

Fig. 11: Statistical object recognition (2D): image, segmentation objects of 2D points, and localized object

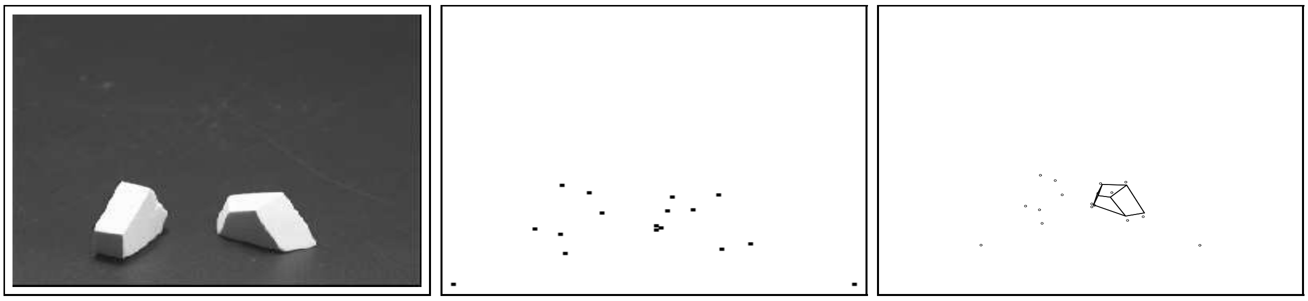

Fig. 12: Statistical object recognition (3D): 2D image, segmentation objects of 2D points, and localized 3D object

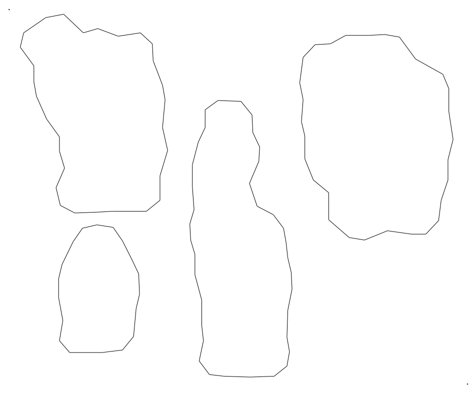

Fig. 10: Polygon segmentation result

functions which are parameterized regarding the pose. Both, learning and localization of objects is formalized as a maximum–likelihood problem. Results for 2D can be seen in Fig. 11. The center image shows a segmentation object consisting of vertices which are special cases of segmentation objects consisting of a point and a set of intersecting lines. 3D statistical object models are generated from different aspects of an object captured with a calibrated camera on a robot ([2]). Input and results of 3D object recognition are shown in Fig. 12.

Similar to the operator hierarchy, optimization is programmed as a hierarchy of classes. Strategies for object recognition reference the abstract interface in the base class and can be simply exchanged — even dynamically during runtime. Extensibility of the system is facilitated as well. Computation time depends on the complexity of the scene and on the number of features. In the 2D case they vary from 10 seconds up to 180 seconds, in the 3D case they are around 90 seconds.

7 Efficiency

The applications in Sect. 6 together with several other programs are now used to judge the use of object–oriented programming for image analysis.

7.1 Code Size

Object–oriented programming does generally not reduce the lines of code compared to conventional implementations. The number of functions is clearly reduced and replaced by classes and methods. The number of methods tends to be larger than functions in conventional solutions. Since the method definitions are bundled by classes, they are mostly better documented.

In traditional systems, programmers define only those functions which are actually needed. Class designers tend to provide a complete interface; often they can be forced to do so by abstract base classes which require the redefinition of certain methods. This tendency increases the number of code lines but clearly enhances reusability by a broader range of possible applications. To give an example from *ἵππος*, the method for rotating geometric objects is declared in the abstract base class and passed to all derived classes by inheritance; this method was required only in few applications, resp. classes; all the other classes just inherit the interface and can define the method, if needed.

The design goal of modularity is almost guaranteed by the use of classes. The average number of methods in 98 important *ἵππος* classes is 8.1; only those methods which are not required by NIHCL are counted. The number of code lines per method does not exceed the length of a page.

A comparison of the conventional implementation of ERNEST and the new object–oriented implementation outlined in Sect. 5.4 shows a reduction of about 50% for the new system [70].

7.2 Memory Requirements

Object–oriented programming in C++ clearly increases the size of the running programs. The use of flexible tools like NIHCL will always result in the problem that code will be included in the executable program which will not be needed in the particular application. Due to the principal undecidability of the problem whether a function will be actually called in a program, the linker will have to include some virtual functions which will not be actually called in the execution of the application. For experimental system, dynamic link libraries (shared libraries) partially circumvent this problem. For real applications, the user may of course manually select only those functions and libraries for the linker which will actually be used. On the other hand, the increase of the executable's size is relatively small compared to the absolute size, especially if graphics routines are used.

Typical program size for a single operator in Animals is 420 KByte (e.g. a Sobel operator) when static libraries for NIHCL are used. When several operators are combined into one program, or when additional objects are required, the size increases only little. For example, a program using the camera object and applying either a Sobel operator or another edge detector and a line tracking algorithm on the captured image has around 600 KByte. For comparison, dynamic linkage of a simple program with X11 results in a minimum of 600 KByte for a program.

7.3 Run Times

As mentioned in Sect. 6.1, Animals can be used in real time applications [12]. This is one of the ultimate goals of image analysis in today's systems. Time consuming low–level operations can run with zero overhead after some precaution on the matrix classes.

Efficiency is one of the design goals for the operator hierarchy (Sect. 5.2). No measurable difference in run time behavior could be observed in the comparison of operator objects and function calls.

Actual limitations in real time processing result from limitations of the hardware, not from the software. It is still difficult to get a color image of reasonable size (512^2) into main memory in video rate (25 Hz), and simultaneously have a low load on the CPU. This would enable active vision techniques to process images selectively.

7.4 Software Life Cycle

All source code in Animals is under revision control. Using the versioned files, the following estimations were possible, using chain code classes as an example. The class `Chain` for chain code representation of lines was initially specified in 1988; a student's theses was finished on this topic in 1989 [45]. By that time, this class was already being used by several applications. The software was almost unchanged until 1992, when a complete redesign of the internals was done in order to maximize run time efficiency. Existing software which only uses the class did not have to be changed, except for few (mechanical) substitutions. In 1993, the *ἵππος*–hierarchy was restructured and the OOPS software was substituted by its successor NIHCL [25]. Around 120 classes had to be adapted to the new package; the work required was about 0.4 man years for the complete change. This example demonstrates how essential parts of a software system are still usable after 6 years; this is unusual in a continuously changing and expanding system with several dozens of implementors. This is mostly due to object–oriented programming.

The history of Animals also revealed how strategic decisions for object–oriented software design can influence software maintenance considerably. In 1989, we decided to extend OOPS with machine independent binary storage facilities which are essential for image processing in a distributed environment. We did so by a modification of the OOPS source code. These changes were of course lost when the NIHCL update arrived. For NIHCL, we also provided machine independent binary storage; now we used inheritance from existing NIHCL classes. We had to sacrifice a little runtime efficiency but *no* changes of the distributed source code were required, i.e., a further update will no longer require changes of the own interfaces.

8 Conclusion and Outlook

Object–oriented programming is now an established tool for image processing and analysis. It is no longer a research item of its own as it was in [46] or [22]. The number of C++ applications for image processing is large. An increasing number of tools for object–oriented programming will be available as class libraries. These can be reused in image processing when new classes are derived from image processing classes and tool classes by multiple inheritance.

We presented a system for image analysis which makes extensive use of object–oriented programming. The general framework is provided in an environment for pattern analysis (PUMA) which is useful for image and speech analysis and contains the object–oriented implementation of a semantic network. Data for image processing is represented in *ἵππος* classes. An image analysis system (Animals) uses operator classes to compute data represented in *ἵππος*. We showed that this framework can be successfully applied to real world problems, including 3D and real time processing.

Acknowledgements

The authors wish to express their thanks to the members of the image processing group of the chair and of the "Forschungszentrum für Wissensbasierte Systeme, Erlangen" for their ideas and software contributions to the system.

SUN & HP are trademarks of SUN microsystems resp. Hewlett Packard.

References

[1] R. B. Arps and W. K. Pratt, editors. *Image Processing and Interchange: Implementation and Systems*, San Jose, CA, 1992. SPIE, SPIE Proceedings 1659.

[2] R. Beß. Kalibrierung einer beweglichen, monokularen Kamera zur Tiefengewinnung aus Bildfolgen. In W. G. Kropatsch and H. Bischof, editors, *Tagungsband Mustererkennung 1994*, volume 5 of *Informatik Xpress*, pages 524 – 531, Berlin, 1994. Springer.

[3] G.M. Birtwistle, O. Dahl, B. Myrhang, and K. Nygaard. *Simula Begin.* Auerbach Publ. Inc., Philadelphia, PA, 1983.

[4] Ch. Blum. Design principles and applications of ISO/IEC's image interchange facility (IIF). In *ImageCom Conferenc*, pages 160–165, Bordeaux, 1993.

[5] Ch. Blum and G. R. Hofmann. ISO/IEC's image interchange facility (IIF). In Arps and Pratt [1], pages 117–129.

[6] T. Butler and P. Krolak. An overview of the Programmer's Imaging Kernel System (PIK) proposed standard. *Computers and Graphics*, 15(4):465–472, 1991.

[7] I. C. Carlsen and D. Haaks. IKS^{PFH} — concept and implementation of an object–oriented framework for image processing. *Computers and Graphics*, 15(4):473–482, 1991.

[8] A. F. Clark. An international standard for image processing and interchange. *PAMI Technical Committee Newsletter*, 14:5–6, 1991.

[9] A. F. Clark. Image processing and interchange — the imaging model. In Arps and Pratt [1], pages 106–116.

[10] P. Coad and E. Yourdon. *Object-oriented analysis.* Prentice Hall, Englewood Cliffs, NJ, 2nd edition, 1991.

[11] B.J. Cox. *Object oriented Programming / An Evolutionary Approach.* Addison-Wesley, Reading, MA, 1986.

[12] J. Denzler, R. Beß J. Hornegger, H. Niemann, and D. Paulus. Learning, tracking and recognition of 3D objects. In V. Graefe, editor, *International Conference on Intelligent Robots and Systems – Advanced Robotic Systems and Real World*, volume 1, pages 89–96, 1994.

[13] J. Denzler and H. Niemann. A two-stage real time object tracking system. In Pavešić et al. [54].

[14] J. Denzler and D. W. R Paulus. Active motion detection and object tracking. In ICIP 94 [33], pages 635–639.

[15] M.R. Dobie and P.H. Lewis. Data structures for image processing in C. *Pattern Recognition Letters*, 12:457–466, 1991.

[16] W. Eckstein, G. Lohmann, U. Meyer–Gruhl, R. Riemer, L. Altamirano Robler, and J. Wunderwald. Beuntzerfreundliche Bildanalyse mit $\mathcal{HORUS}$: Architektur und Konzepte. In S. J. Pöppl and H. Handels, editors, *Mustererkennung 1993*, pages 332—339, Berlin, 1993. Springer.

[17] G. Enderle, K. Kansy, and G. Pfaff. *Computer Graphics Programming, GKS: The Graphic Standard.* Springer, Berlin, 1984.

[18] H. Faasch. Konzeption und Implementation einer objektorientierten Experimentierumgebung für die Bildfolgenauswertung in ADA. PhD thesis, Hamburg, 1987.

[19] V. Fischer. Parallelverarbeitung in einem semantischen Netzwerk für die wissensbasierte Musteranalyse. Technical report, Dissertation, Technische Fakultät, Universität Erlangen–Nürnberg, Erlangen, 1995.

[20] D. Le Gall. MPEG: A video compression standard for multimedia applications. *Communications of the Association for Computing Machinery*, 34(4):47–58, April 1991.

[21] P. Gemmar and G. Hofele. *Empfehlung für ein Ikonisches Kernsystem IKS.* FIM Informationsverarbeitung und Mustererkennung), Karlsruhe, 1989. with contributions of: L. Dreschler-Fischer, H. Faasch, D. Haaks und D. Paulus, .

[22] P. Gemmar and G. Hofele. An object oriented approach for an iconic kernel system IKS. In ICPR 90 [34], pages 85–90.

[23] P. Gemmar and G. Hofele. Design of an

iconic kernel system. *Computers and Graphics*, 15(4):483–493, 1991.

[24] A. Goldberg and D. Robson. *Smalltalk-80: The Language and its Implementation.* Addison-Wesley, Reading, MA, 1983.

[25] K. E. Gorlen, S. Orlow, and P. S. Plexico. *Data Abstraction and Object–Oriented Programming in C++.* John Wiley and Sons, Chichester, 1990.

[26] A. R. Hanson and E. M. Riseman, editors. *Computer Vision Systems.* Academic Press, New York, 1978.

[27] A. R. Hanson and E. M. Riseman. VISIONS: A computer system for interpreting scenes. [26], pages 303–333.

[28] R. M. Haralick and V. Ramesh. Image understanding environment. In Arps and Pratt [1], pages 159–167.

[29] M. Harbeck. *Objektorientierte linienbasierte Segmentierung* . Dissertation, IMMD 5 (Mustererkennung), Universität Erlangen–Nürnberg, Erlangen, 1996.

[30] J. Hornegger and H. Niemann. A Bayesian approach to learn and classify 3–D objects from intensity images. In *Proceedings of the 12th International Conference on Pattern Recognition (ICPR)*, pages 557–559, Jerusalem, October 1994. IEEE Computer Society Press.

[31] J. Hornegger and H. Niemann. The missing information principle in computer vision. In Pavešić et al. [54], pages 113–126.

[32] J. Hornegger, H. Niemann, D. W. R. Paulus, and G. Schlottke. Object recognition using hidden Markov models. In E. S. Gelsema and L. N. Kanal, editors, *Pattern Recognition in Practice IV: Multiple Paradigms, Comparative Studies and Hybrid Systems*, volume 16 of *Machine Intelligence and Pattern Recognition*, pages 37–44, Amsterdam, June 1994. Elsevier.

[33] *Proceedings of the International Conference on Image Processing (ICIP)*, Austin, TX, USA, November 1994. IEEE Computer Society Press.

[34] *Proceedings of the 10th International Conference on Pattern Recognition (ICPR)*, volume 2, Atlantic City, 1990. IEEE Computer Society Press.

[35] S. Impedovo, editor. *Progress in Image Analysis and Processing III, Proceedings 7th International Conference on Image Analysis and Processing*, Bari, Italy, 1994. World Scientific.

[36] International standard 12087, image processing and interchange. Technical report, International Standards Organization, Genf, CH, to appear 1994.

[37] T. Kawai, H. Okazaki, K. Tanaka, and H. Tamura. VIEW–station software and its graphical user interface. In Arps and Pratt [1], pages 311–323.

[38] P. Koller. Computation of range images from color–stereo–images by 'simulated annealing'. In Pavešić et al. [53], pages 119–130.

[39] P. Levi. Ikonisches Kernsystem. *Robotersysteme*, 1:172–178, 1985.

[40] M. A. Linton, J. M. Vlissides, and R. P. Calder. Composing user interfaces with InterViews. *IEEE Computer*, pages 8–22, 1989.

[41] J. Mundy, T. Binford, T. Boult, A. Hanson, R. Veveridge, R. Haralick, V. Ramesh, C. Kohl, D. Lawton, D. Morgan, K Price, and T. Strat. The image understanding environments program. In *Proc. of the DARPA Image Understanding Workshop*, pages 185–214, Hawaii, Jan. 1992.

[42] H. Niemann. *Pattern Analysis and Understanding.* Springer, Heidelberg, 1990.

[43] H. Niemann. Interpretation of image sequences. In W. Zamojski and D. Caban, editors, *Proceedings of the 5th school computer vision and graphics*, pages 57–72. Univ. Wrocław, Wrocław, 1994.

[44] H. Niemann, G. Sagerer, S. Schröder, and F. Kummert. Ernest: A semantic network system for pattern understanding. *IEEE*

Transactions on Pattern Analysis and Machine Intelligence (PAMI), 9:883–905, 1990.

[45] M. Oestreich. Linien als Objekte für die Bildverarbeitung. Student's thesis, IMMD 5 (Mustererkennung), Universität Erlangen–Nürnberg, Erlangen, 1988.

[46] D. Paulus. *Objektorientierte und wissensbasierte Bildverarbeitung "Object–oriented and knowledge based image processing.* Vieweg, Braunschweig, 1992.

[47] D. Paulus and H. Niemann. Object–oriented programming for image analysis. In J. Menon, editor, *Current Topics of Pattern Recognition Research*, volume 1 of *Research Trends*, pages 185–204. India, 1994.

[48] D. Paulus, A. Winzen, F. Gallwitz, and H. Niemann. Object–oriented knowledge representation for image analysis. In Pavešić et al. [54], pages 37–54.

[49] D. Paulus, A. Winzen, and H. Niemann. Knowlege based object recognition and model generation. In Donald W. Braggins, editor, *Proceedings Europto 93, Computer Vision for Industry, München*, pages 382–393, Bellingham, WA, 1993. SPIE, SPIE. Proceedings 1659.

[50] D. W. R. Paulus. Object oriented image segmentation. In *Proc. of the* 4^{th} *Int. Conf. on Image Processing and its Applications*, pages 482–485, Maastrich, Holland, 1992.

[51] D. W. R. Paulus. Object–oriented stereo analysis. In Pavešić et al. [53], pages 131–150.

[52] D. W. R. Paulus and H. Niemann. Iconic–symbolic interfaces. In Arps and Pratt [1], pages 204–214.

[53] N. Pavešić, H. Niemann, and D. Paulus, editors. *Proceedings of the German–Slowenian Workshop on Image Processing and Stereo–Analysis.* Arbeitsberichte des IMMD der Universität Erlangen–Nürnberg, Band 26/1, Erlangen, 1993.

[54] N. Pavešić, H. Niemann, D. Paulus, and S. Kovaćić, editors. *3–D Scene Acquisition, Modeling and Understanding, Proceedings of the Second German–Slovenian Workshop*, Ljubljana, Slovenia, June 1994. IEEE Slovenia Section.

[55] C. A. Poynoton. An overview of TIFF 5.0. In Arps and Pratt [1], pages 150–158.

[56] W. K. Pratt. Overview of the ISO/ICE programmer's imaging kernel system application program interface. In Arps and Pratt [1], pages 117–129.

[57] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C – The Art of Scientific Computing.* Cambridge University Press, New York, 1990.

[58] J. R. Rasure and M. Young. Open environment for image processing and software development. In Arps and Pratt [1], pages 300–310.

[59] E.M. Riseman and A.R. Hanson. A methodolgy for the development of general knowledge–based vision systems. In C. Torras, editor, *Computer Vision, Theory and Industrial Applications*, pages 293–336. Springer, Berlin, Heidelberg, New York, 1992.

[60] Markku Sakkinen. On the darker side of C++. In *Object-Oriented Programming Systems, Languages and Applications*, pages 162–176. ACM Press, 1988. Conference Proceedings OOPSLA.

[61] H. Sato, H. Okazaki, T. Kawai, H. Yamamoto, and H. Tamura. The view-station environment: Tools and architecture for a platform-independent image-processing workstation. In ICPR 90 [34], pages 576–583.

[62] S. Schröder. *Integration einer Wissenserwerbkomponente in eine Systemumgebung für die Musteranalyse.* Reihe 10: Informatik / Kommunikationstechnik. VDI Verlag, Düsseldorf, 1990.

[63] B. Stroustrup. *The C++ Programming Language.* Addison-Wesley, Reading, Mass., 2^{nd} edition, 1991.

[64] Sun OS 4 Manuals, Network Programming,

Part 2, Mountain View, CA. *RPC eXternal Data Representation Standard: Protocol Specification*, Revision B, 1986.

[65] M.J. Swain and M. Stricker. Promising directions in active vision. Technical Report CS 91-27, University of Chicago, 1991.

[66] H. Tamura et al. Design and implementation of spider - a transportable image processing software package. *Computer Vision, Graphics and Image Processing*, 23:273–294, 1983.

[67] G. Wallace. Overview of the JPEG (ISO/CCITT) still image compression standard. In *Electronic Image Science and Technology*, pages 97–108. SPIE Proceedings 1244, Santa Clara, CA, Feb. 1990.

[68] T. D. Williams and R. R. Kohler. Environment for image understanding development. In Arps and Pratt [1], pages 341–349.

[69] A. Winzen. Efficient methods for hypothesis verification paradigms. In *Proceedings of the* 11*th* *International Conference on Pattern Recognition (ICPR)*, pages 558–561, The Hague, Netherlands, August 1992. IEEE Computer Society Press.

[70] A. Winzen. Automatische Erzeugung dreidimensionaler Modelle für Bildanalysesysteme. Technical report, Dissertation, IMMD 5 (Mustererkennung), Universität Erlangen–Nürnberg, Erlangen, 1994.

[71] A. Winzen. Matching of 3D–lines for automatic model–generation. In Impedovo [35], pages 607–614.

[72] A. Winzen and H. Niemann. Matching and fusing 3d–polygonal approximations for model–generation. In ICIP 94 [33], pages 228–232.

[73] A. Winzen and H. Niemann. Automatic model–generation for image analysis. In W. Strasser and F. Wahl, editors, *Graphics & Robotics*. Springer, Berlin, to appear 1994.

[74] A. Winzen, D. Paulus, H. Niemann, and V. Fischer. Semantische Netze für die Bildanalyse: Objektorientierte Realisierung mit paralleler Kontrolle. In H. Wedekind, editor, *Verteilte Systeme*, pages 371–386. BI Wissenschaftsverlag, Mannheim, 1994.