An Interactive CAD-Based Vision System

Behrouz M-Rouhani,
Anthony D. Worrall, James A.D.W. Anderson
Intelligent Systems Group, Department of Computer Science
The University of Reading, Whiteknights, Reading RG6 6AY
United Kingdom
James.Anderson@reading.ac.uk

Abstract

We report a preliminary implementation of an interactive system that enables a draftsperson using only the standard features of a commercial CAD package (AutoCAD) to construct a model-based vision program. Some of the information needed by the model-based vision program is computed automatically from the drawing, but the draftsperson can also provides guidance on expected image-feature types by drawing the corresponding parts of the model on named CAD-layers and can indicate the visibility of arbitrary parts, such as expected shadow lines, by connecting them to modelled parts with invisible construction lines. Our system, running on a workstation, converts small drawings to computer vision programs in approximately 5 seconds, though an additional 35 seconds is used on networking, windowing, and software initialisation.We argue that interactive CAD-based vision compilers are more useful than contemporary batch-mode CAD-based vision compilers and conclude that the emergence of the international standard STEP will support the development of more competent CAD-based vision compilers of all kinds.

1 Introduction

It has long been the goal of computer vision to exploit CAD software, though the term 'CAD-based vision' seems to have been coined as late as 1987 [8] and is generally taken to refer to visual processing using representations that *might* be generated by a CAD package (p. 107) [10]. The majority of CAD-based vision systems so far reported [15][16][17] do not use commercial CAD packages, though [5] is a notable exception. We now report the use of a commercial CAD package which has been modified so that it generates model-based vision programs interactively. The draftsperson prepares a drawing using AutoCAD and then selects customised menu items to convert the drawing into a model-based vision program and test it on a database of video images. This is done in three stages. Firstly, the in-core drawing is saved to disc using AutoCAD's drawing interchange format (DXF). Secondly, a translator converts the DXF file into our proprietary Primitive File Format (PFF). Finally, our model-tracking program reads in the video images and PFF data, then the draftsperson initiates tracking, which continues automatically.

In the subsequent sections we give the first published account of our facet modeller and Primitive File-Format (PFF) that were used in numerous published papers in the past 10 years. We describe the AutoCAD facilities that can be used to prepare drawings in sufficient detail to allow the reader to construct models in AutoCAD that could be processed by our model-based vision programs. We conclude with a general discussion of the role of commercial CAD software in CAD-based vision and recommend consideration of the emerging *de jure* STandard for Exchange of Product model data (STEP) [12][9] as a means of integrating CAD and computer vision.

2 Facet Modeller

In this section we describe the over-all structure of the facet modeller and explain the role of the Primitive File-Format (PFF) which is used to describe a single geometrical object. The data structures which are used to combine primitives into models are not described in detail. The AutoCAD interface does, however, generate all of the required data structures automatically. In the remainder of this section all words in *italics* are symbols taken from the PFF grammar in Section 8.1.

The facet modeller supports near real-time evaluation of models. It operates by evaluating evidence for straight *lines* in one or more images of an object [20]. A *line* can be an *edge* (first difference) or *bar* (second difference) in an image. The scores for each *line* of the model are combined to give a score for the model in the current pose (position and orientation). This can be used in a hypothesize-test-refine strategy to locate, classify, and track vehicles [19].

The modeller is designed in a similar way to an Additive Constructive Solid Geometry [2][3]. A tree structure is used with nodes representing coordinate frames and arcs representing transformations between coordinate frames. These transformations can be arbitrary affine transformations, but are usually rigid transformations. Each node in the tree contains a pointer to its parent, a list of all of its children, the transformation to the parent coordinate frame, and a cache of the transformations to the world coordinate frame which is at the root of the tree. The leaf nodes can describe either *primitive* solids or cameras.

A *primitive* used by the modeller can be tailored to the object or part of the object under consideration, it need not be a simple regular solid. The *primitive* has three main components, a matrix of all the 3D *points* used in the primitive, a list of *lines* used by the evaluator, and a list of *facets* (bounded planes) which are used during hidden line removal.

A number of hidden line removal strategies are supported. In the case of an isolated, *convex primitive* the visibility of a *line* can be decided by considering just a logical operation on the visibility of the *facets* with which it is associated. For example, an *extremal* line between facets is visible if exactly one facet is visible, a *fold line* between *facets* is visible if either *facet* is visible, a *crease line* between *facets* is visible if and only if both *facets* are visible, a *mark line* on a *facet* is visible if that *facet* is visible, and a *wire line* is visible regardless of any *facet*. The visibility of a *facet* is determined by the sign of the dot product between the outward pointing *normal* to the plane and the view direction (optical axis) of the camera. (This is done in camera coordinates where, by convention, the optical axis lies on one of the principal axes.) In the case of *concave* and/or multiple objects these heuristics are not sufficient and clipping of these ‘visible’ lines is required. This is done using the Cyrus-Beck parametric line clipping algorithm with the Liang-Barsky extension. A full discussion of this extended algorithm can be found in [14]. The extended algorithm clips lines against convex polygons in the viewing plane and in the polygon's plane in 3D. Hence *concave facets* are also described by a *tessellation* of convex polygons. The modeller also supports a mode in which the current line clipping is kept, even when the model is moved. This is used during pose refinement when only small changes are made to the pose which do not have a great effect on the visibility of *lines*. The line clipping algorithms can be swapped at run time.

The definition of a camera, in a node, contains the intrinsic parameters of the camera. These are used with the world transformation stored in the node to compute a perspective transformation from world coordinates to normalised camera coordinates. Line clipping is done in this normalised coordinate frame with orthogonal projection onto the image plane. Each camera also contains a tree structure that mirrors the tree structure of the 3D world tree, but in normalised camera coordinates; it is used to represent the 2D structure of the projected world. Each node in this 2D tree corresponds to a node in the 3D tree and there are cross links in both directions to avoid tree searches. Each node also contains bounding box information for the sub-tree which is used to determine if a *line* needs to be clipped against the sub-tree.

A full model describes: the geometry, image features, and hidden-line removal heuristics for an object or objects that move in a scene; the fixed geometry of the scene; and the variable geometry of a camera or cameras. Full hidden-line removal can be computed using the extended Cyrus-Beck algorithm, but the heuristics allow faster, conservative hidden-line removal in special cases. An important special case is during pose-refinement on one image, where no further hidden line removal is required until a the next image is processed.

3 The AutoCAD and Facet Modeller Interfaces

AutoCAD's user interface provides a drawing surface with mouse interaction, a text surface, a command line, and pop-up menus. All of the interaction modes can be programmed in C or AutoLisp, a proprietary variant of Lisp. We extended the interface using AutoLisp.

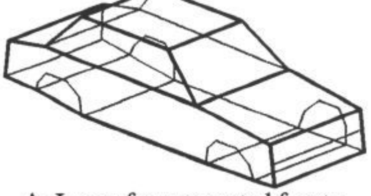
The standard AutoCAD user interface is used to read and write files, but an additional menu button has been provided that automatically carries out the translation from DXF to PFF and calls the facet model interface which in turn calls the model-tracker. The facet model interface is implemented in Pop11 [1] with call out to C. It is a wimp interface that allows the user to import sequences of video images from disc, to import models in PFF, to specify the initial pose of a model, and to begin tracking.

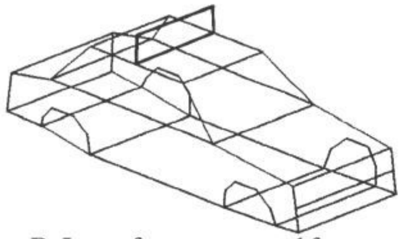
The whole process from initiating a translation to the start of tracking takes about forty seconds on a networked Sparcstation 1, of which twenty seconds is used on network communication, and fifteen seconds is used initialising the visual tracking program, only five seconds is needed to export a drawing in DXF and convert it to PFF. The networking portion and some of the time to initialise windows and the tracking program would be unnecessary had we a license that would permit tight integration of our code and AutoCAD. This would give an edit-compile-test cycle of about fifteen seconds.

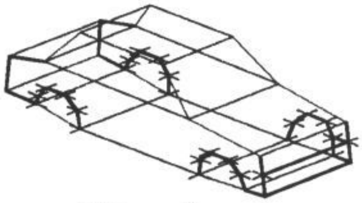
4 Translation from DXF to PFF

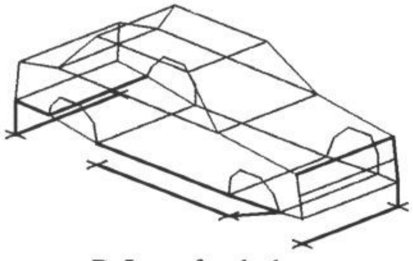
DXF is a *de facto* standard, but, like other CAD data exchange standards, it does not provide a rigorous definition of drawings. It is an ASCII format which was designed to be parsed by Fortran programs and has the unusual property that some symbols are context sensitive (they index other symbols that may occur earlier or later in the code). This makes it somewhat difficult to design an efficient DXF parser. We describe here just the AutoCAD facilities that draftspeople should use, if they wish their drawings to be capable of generating model-based vision programs using our system.

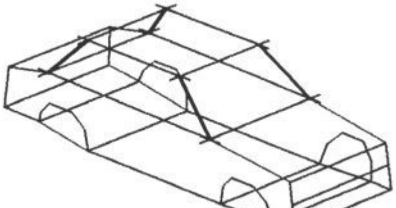
AutoCAD, along with many CAD systems, allows arbitrary sets of components to be collected together in drawing 'layers'. The basic drawing of a saloon car is shown on Layer A, below, and is repeated on the other layers only to show the reader how the parts on those layers, shown in bold, are related to the overall drawing. Layers A and B contain only objects drawn with AutoCAD's PFACE command. Objects on layer A must be planar facets, connected along their edges to produce an orientable, non-self intersecting 2D manifold. The facet edges on this layer are treated as image edges (maxima in the first image difference). Here the visible facet edges are shown emboldened. Layer B shows an unconnected facet which need not be manifold. This layer is intended for rendering background details such as road markings, but is used here to put a sign board on the car roof. Layers C and D contain only polylines drawn with AutoCAD's 3DPOLY command. These polylines are connected to facets. The first three edges of a polyline are nominally invisible construction edges that are used to tie the feature to a facet, the remaining edges, shown with crosses on the end points, are nominally visible. Layer C is intended for marks on a facet and layer D is intended to indicate the approximate position of shadows that occur under typical lighting conditions. In daylight scenes the light is always above. Layer E marks those edges in any layer that are to be treated as image bars (maxima in second image difference).


A: Layer for connected facets: image edges


B: Layer for unconnected facets


C: Layer for marks


D: Layer for shadows


E: Layer for image bars

The above layered drawing is converted automatically to a PFF model for use by our image tracking program, or by any of the model-based vision programs that we have published in recent years. The translation proceeds in a number of steps. The common edges of layer A (edges of adjacent PFACE objects) are found and are categorised into convex or concave edges, this information is needed to satisfy the hidden-line removal heuristics. It is important that these edges are consistently drawn anti-clockwise as seen from outside the object. This ordering is needed to recover the sense of the outward pointing surface normal. (Please note that the mirroring instructions in AutoCAD do not preserve edge ordering in facets.) The marks (polylines) on layers C and D are extracted and the first two edges are examined to see which two facet edges they are common with. This uniquely identifies the facet they are to be considered a part of for hidden-line removal. The third edge is ignored, it is used to place the remaining edges in space. Only the remaining edges are stored in PFF along with the name of the facet they belong to. The use of shadow edges is a heuristic that works adequately for tracking motor cars because they are close to the ground and the light source is always above. Shadows provide very strong image contrast so are excellent image features when they can be accurately predicted - as here. The unconnected facets on layer B are duplicated in the PFF as two coincident facets with opposite surface normals. This is a more efficient way of modelling essentially laminar objects than constructing their manifold. All of the edges so far discovered are treated as image edges (maxima in first difference), unless they also occur in layer E, in which case they are treated as image bars (maxima in second difference). Thus the default image properties, geometry, and topology of models are given by layers A-D and special image properties are given by layer E. in practice we would need to program the allocation of layers in a drawing to our layers because DXF does not adequately specify the name space of layers. Once this relationship has been established, however, the conversion is automatic.

The use of layers and (invisible) construction lines is a natural method for *CAD* draftspeople and is effective for encoding geometrical, topological, and semantic information. However, this does require a draftsperson skilled in computer vision.

5 Results

We are able to draw models in AutoCAD that are identical to the hand-constructed drawings used in our previous research. Though we can, of course, draw and test models much more quickly than previously - in a matter of seconds. Three successful trackings are shown in Section 9 where the superimposed model-outlines have been thickened by hand so that they reproduce well. Here only two models are used, a saloon and a hatchback. The english saloon model has, however, been scaled by a factor of 1.2 so that it matches the larger german saloon car!

6 Discussion

Previous research in CAD-based vision has concentrated on the off-line compilation of exhaustive visual search strategies. This leads, quite naturally, to the development of batch compilers with long execution times. Our previous work on CAD-based vision was no exception [4]. By contrast, we have now demonstrated an interactive system that converts drawings to model-based vision programs. In this pilot implementation the vision program is actually a hand-written interactive system that allows the tracking of rigid models. This model-evaluation engine was used in previous research [18], so we can compare tracking of AutoCAD models with hand-written models and models created using structure from symmetry design tools [6]. Our initial experience is that all of these models are effective at tracking.

Our CAD-based vision system is interactive, so the draftsperson has immediate feedback on the performance of the vision programs. The performance is immediately related to the change in the drawing that has just been made. By contrast, in a traditional batch-mode CAD-based vision system, many changes would be made in the course of a working day which might be compiled overnight. This would make it much more difficult to discover the effect of individual changes. With a batch-mode CAD-based vision system the draftsperson requires far greater knowledge of computer vision theory. Whereas, we suppose, a draftsperson with modest experience of computer vision could use our interactive CAD-based vision system.

We chose AutoCAD because it is a popular package which is widely used for engineering, architectural design, and many other disciplines. The user community is well developed with convenient internet access and is used to dealing with the many add-on packages that exist for AutoCAD. Thus AutoCAD seemed a natural choice for a pilot study in an add-on CAD-based vision system. Also, AutoCAD was available to us at no cost, which is an important consideration in unfunded research.

However, as we discussed in Section 4, AutoCAD and DXF provide inadequate topological information. This problem is quite general in industry standard interchange formats and has been addressed elsewhere [13]. The favoured solution is to provide the designer with customised dialogues that ask for topological information or constrain the input of a drawing so that topological data can be extracted from it. The topological information can then be saved by, for example, including it as a comment or text item in the interchange file. We wish to avoid this arbitrary approach, if at all possible, and provide entirely automatic compilation of vision programs from CAD drawings. Where we cannot avoid special user input we wish to convey the special information in a well formulated way, such as by using CAD-layers and construction lines, that can be represented by the standard interchange formats.

We have examined the emerging standard ISO 10303 STandard for Exchange of Product model data STEP [12] [9]. It provides very broad coverage of geometrical, topological, and product data. This is achieved by separating the description of the physical properties of an object from its abstract properties. This separation is maintained by a high-level language EXPRESS that uses a wide range of standard protocols to access and exchange data. By defining Applications Protocols (APs) in EXPRESS, it becomes possible to represent a great deal of user-defined data in a STEP file, which can then be exported to a wide variety of systems. One interesting result of this practice would be the possibility for individual user communities, such as the computer vision communities, to define their own STEP implementations - the so called Disposable Applications Protocols (DAPs) [12]. This could open the way to greater interchange of results within the vision communities and the publication of requirements (DAPs) that CAD vendors might come to support.

STEP would seem to offer adequate geometrical and topological information, in which case we should be able to implement a second pilot system with greater generality.

In future research, for which we have funding, we intend to take advantage of the sound geometrical modelling facilities in a more expensive commercial CAD package that supports STEP. We will use the package to input drawings in a number of formats and convert them to triangulated meshes. We believe that the simple geometry and topology of a triangulated mesh will allow us to compute image features efficiently, from which we can generate appropriate PFF models. For this reason, triangulated meshes would seem to be a good choice for a CAD-based vision representation. In addition, triangular meshes are very widely available in commercial CAD packages and all existing solid-object representations can be converted to them.

High performance commercial CAD packages also have the advantage that they can record the material from which parts are made and surface finishes that are applied to them. Indeed, arbitrary product modelling data can be associated with a part and can be indexed syntactically from a database or spatially from a drawing (including 2D perspective drawings). This should provide the ability to search for syntactic cues (gross image features) to index models or to search spatially for alternative surface properties that might explain an unexpected image feature that occurs on an otherwise well matched object. With full product data available, we can also expect to propagate visual constraints between parts of a model. In the case of cars, we might propagate the colour of any body panel to all others, but retain the default properties of windows, headlights, and plastic trim. There is also the possibility of using the graphics engines within high performance CAD packages to pre-compute within-object hidden line removal, thus speeding up our model-evaluation engine. We might then achieve video rate tracking on a workstation.

7 Conclusion

This pilot study has met some of the requirements for an interactive CAD-based vision system. It demonstrates the speedy and effective transfer of drawings from a commercial CAD package, AutoCAD, to a model-based vision system. This is a step toward closing the gap between the academic community and industry vendors as discussed in [16] and demonstrates further de-coupling of model-building and model-evaluation in a vision program. However, it has exposed a fundamental weakness in the *de facto* DXF standard. Namely, the absence of explicit topological information. This issue has been addressed elsewhere by using customised drawing-interfaces to deliver non-standard data [11][13], but we believe that the emerging *de jure* standard STEP will address the need for topological information as well as enabling local conventions for describing the finish and surface colouring of objects. This, in turn, will support the development of more effective CAD-based vision systems. There is also the possibility of feeding our experience back to the developers of STEP and CAD systems, so that CAD-based vision becomes a commercially supported application.

We believe that we have described our CAD-based vision system in sufficient detail that the reader can prepare DXF drawings of objects to test our vision programs. In the longer term we hope to promote a STEP Disposable Application Protocol (DAP) for model-based computer vision and we invite the reader to consider the requirements for a DAP in other areas of computer vision research.

8 Appendix A - PFF Format

The Primitive File Format (PFF) is our proprietary format for describing an isolated, geometrical object. PFF is designed so that it can be parsed by Pop11 [1]. Strictly speaking, PFF can be parsed from any compilation unit, not necessarily from a disc file as its name might suggest. In Section 8.1 *real* denotes any Pop11 real number and *string* denotes any word that is itemisable by the Pop11 itemiser using only white space as a separator. Pop11 comments can be included in PFF data, but other items would cause side effects and/or errors.

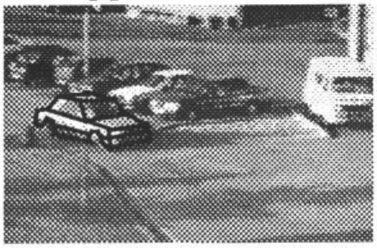
An outward surface normal is given by three points in the order p_1, p_2, p_3 and is computed as $(p_1 - p_2) \times (p_1 - p_3)$. All of the remaining semantics are described in the paper, except for 'Pos' which gives the coordinates of a point, 'Combine' which takes a weighted average of two points, and 'Scale' which scales the coordinates of a point.

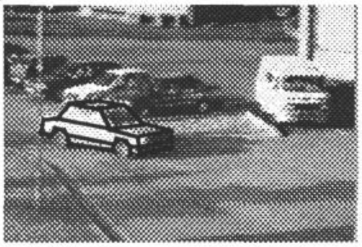
In the grammar which follows there are three meta symbols: '→' denotes a production, '|' denotes alternation, and 'ε' denotes the null symbol. Terminal symbols are denoted in `Courier` and non-terminals in *Times Roman italic*.

8.1 PFF

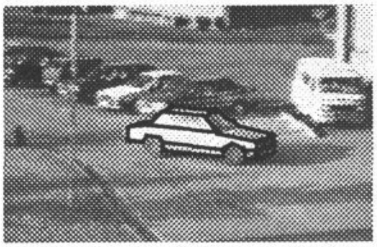
primitive → `[ Primitive` *primitivedata* `]`
primitivedata → *primitvedatum primitivedata* | ε
primitvedatum → *id* | `Type` *type* | `Points [` *points* `]` | `Facets [` *facets* `]` |
        `Lines [` *lines* `]`
points → *point points* | ε
facets → *facet facets* | ε
lines → *line lines* | ε
point → `[ Point` *id pointspecifier* `]`
pointspecifier → `Pos [` *real real real* `]` |
        `Combine [` *real pointname real pointname* `]` |
        `Scale [` *pointname real real real* `]`
facet → `[ Facet` *id normal boundary tessellation* `]`
normal → `[ Normal` *pointname pointname pointname* `]`
boundary → `[` *pointsequence* `]`
tessellation → `[` *pointlist* `]`
line → `[ Line` *id* `Points` *pointlist* `Feature` *feature* `Form` *form*
        `Facet` *facetspecifier* `]`
pointlist → `[` *pointsequence* `]`
facetspecifier → *facetname* | `[` *facetsequence* `]`
pointsequence → *pointname pointsequence* | ε
facetsequence → *facetname facetsequence* | ε
id → `ID` *string*
type → `concave` | `convex`
feature → `bar` | `edge`
form → `crease` | `extremal` | `fold` | `mark` | `wire`
pointname → *string*
facetname → *string*

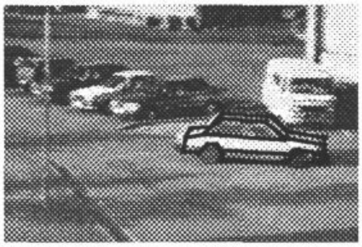
9 Appendix B - Results of Tracking

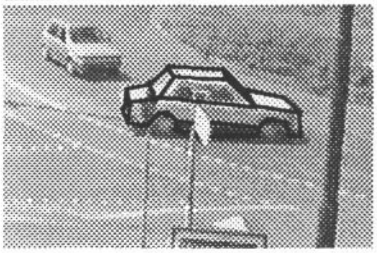

①

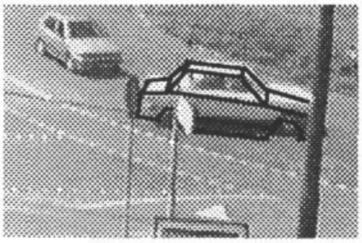
②


Saloon in England

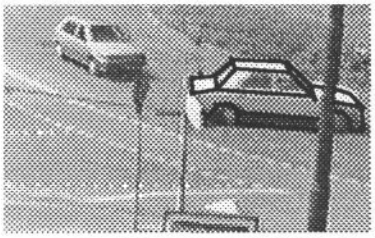

③

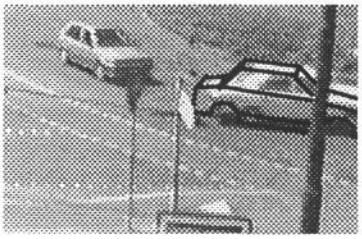
④


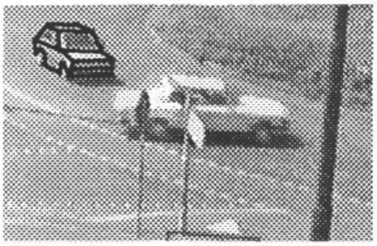

①

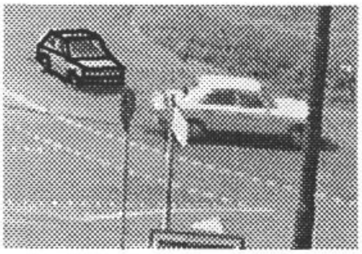
②


Large saloon in Germany

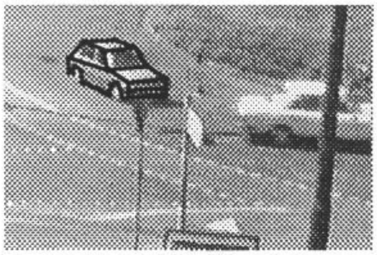

③

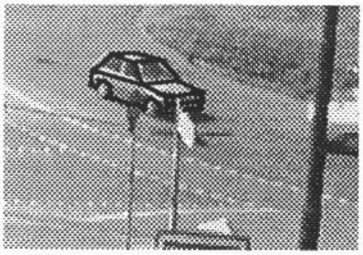
④



①

②


Hatchback in Germany


③

④


References

[1] J.A.D.W. Anderson, ed. *POP-11 Comes of Age: the advancement of an AI programming language* Ellis-Horwood, 1989.

[2] J.A.D.W. Anderson, G.D. Sullivan, & K.D. Baker, 'Constrained constructive solid geometry: a unique representation of scenes' *Proceedings of the fourth Alvey Vision Conference* University of Manchester, England, pp. 91-96, August 1988.

[3] J.A.D.W. Anderson 'Solid Modelling' in *Canonical Description of the Perspective Projections* Ph.D. Thesis, Department of Computer Science, The University of Reading, England, June, 1992.

[4] J.A.D.W. Anderson 'A Demonstration of CAD-Vision Compilation *Research and Development in Expert Systems X* BHR Group, pp. 353-368, December 1993.

[5] F. Arman & J.K. Aggarwal 'Automatic Generation of Recognition Strategies Using CAD Models' *Workshop on Directions in Automated CAD-Based Vision*, Hawii, pp. 124-133, June 1992.

[6] C.I. Attwood, G.D. Sullivan & K.D. Baker 'Model Construction from a Single Perspective View Using Shape from Symmetry, B.M.V.C. 93, Surrey, pp. 155-164, September 1993.

[7] Autodesk Inc. 'Drawing Interchange and File Formats' in *AutoCAD Customization Manual* release 12, pp. 243-284, 1992.

[8] B. Bhanu '*CAD*-Based Robot Vision' pp.13-16 of a special issue of *Computer*, August 1987.

[9] M.S. Bloor, J. Owen '*CAD/CAM* Product data exchange: the next STEP', *CAD*, vol 23, no. 4, May 1994.

[10] K.W. Bowyer 'Introduction to Special Issue on Direction in *CAD*-Based Vision' *CVGIP Image Understanding*, vol. 2, pp. 107-108, March 1992.

[11] O. Camps, L. Shapiro 'PREMIO: An Overview' *Workshop on Directions in Automated CAD-Based Vision*, Hawii, pp. 11-21, June 1991.

[12] L. Celander *Introduktion till STEP* (in Swedish) The Swedish Institute of Production Engineering Research, Göteborg, Sweden, 1994.

[13] P. Flynn, A.K. Jain '*CAD*-Based Computer Vision: From *CAD* Models to relational Graphs', I.E.E.E. Trans. P.A.M.I., vol. 13, no. 2, pp. 114-132, Feb.1991.

[14] J.D. Foley, A. van Dam, S.K. Feiner & J.F. Hughes *Computer Graphics: Principles and Practice* Addison Wesley, 1987.

[15] I.E.E.E. Workshop on Directions in Automated *CAD*-Based Vision, Hawii, June 1991.

[16] S. Negahdaripour & Anil K. Jain *Challenges in Computer Vision: Future Research Directions* NFS Workshop held in Maui, HI, USA, pp. 189-199, June 1991.

[17] Proc. Second *CAD*-Based Vision Workshop, Pensylvania, USA, Feb. 1994.

[18] G.D. Sullivan 'Visual interpretation of known objects in constrained scenes. Phil. Trans. Roy. Soc. Lond. B 337, pp. 361-370, 1992.

[19] A.D. Worrall, G.D. Sullivan & K.D. Baker Model Based Tracking B.M.V.C. 91, Glasgow, pp. 310-318, 1991.

[20] A.D. Worrall, G.D. Sullivan & K.D. Baker 'Advances in model-based traffic vision' B.B.V.C. 93, Surrey, pp. 550-568, 1993.