ELSEVIER

Journal of Systems Architecture 43 (1997) 327–335

JOURNAL OF SYSTEMS ARCHITECTURE

Extended abstract

# VISUAL: An object oriented language for image understanding

S.L. Chen [a], E-ren Chuang [b], W.S. Hsieh [c,*]

[a] *Department of Electrical Engineering, National Sun Yat-Sen University, Kaohsiung, Taiwan, ROC*
[b] *Department of Electrical Engineering, National Kaohsiung Institute of Technology, Kaohsiung, Taiwan, ROC*
[c] *Institute of Computer and Information Engineering, National Sun Yat-Sen University, Kaohsiung, Taiwan, ROC*

**Abstract**

VISUAL is an object-oriented and declarative language for locating objects automatically. The language is designed according to the principle of closeness and the principle of object orientation. A VISUAL program describes the geometric relationships among components of an object. The inference engine of VISUAL opens the corresponding databases and $N \times N$ 2-D maps of the components, then applies chromatographic search for unification. The inference engine outputs a database and a 2-D map of the object which could be a component of another object to be recognized. Hence VISUAL is a tool to build up a hierarchical model-based vision system.

*Keywords:* Object-oriented; Declarative language; Principle of closenes; Chromatographic search

## 1. Introduction

It is generally believed [1] that model-based vision system include three parts: *feature extraction*, *object modeling* and *recognition*. The last part includes a recognition engine. The engine recognizes objects by comparing features extracted from an input image to the object features in models. There are various mechanisms for the recognition engine, as statistical [2] or syntactical [3,4] approach, CAD-based vision system [5,6], and rule-based [7] or prolog-based [8] approach. Different mechanisms rely on different methods for object modeling.

The current trend of developing programming language is to integrate declarative, functional, and object-oriented languages, and to include the word of database and programming [9]. VISUAL is an object-oriented and declarative languages which integrate images and database. Given an image and a description of an object in VISUAL, the object will be automatically located in the image.

This paper is organized as follows: The next section introduces the fundamental designing princi-

* Corresponding author. Email: wshsieh@mail.nsysu.edu.tw.

*PII* S1383-7621/0165-6074(96)00098-7

ples of VISUAL, and how those principle are realized by integrating databases and maps of objects. Section 3 presents a syntax of VISUAL with a working example. In Section 4, the internal execution of VISUAL's inference engine is dissected in details, and the time complexity of the inference engine is analyzed. In Section 5, VISUAL is applied to locate objects in synthesized and read images. Those applications show how to express geometric feature, how to construct hierarchical model-based vision system, and how to perform image understanding tasks in VISUAL.

## 2. Principles in designing VISUAL

We propose a programming, VISUAL, to perform the task of image understanding. VISUAL is designed according to two principles: *principle of object orientation* and *principle of closeness.*

### 2.1. The principle of object orientation

Almost every image understanding task is to locate or to recognize objects. However, the components of these objects are also objects. Since objects are essential elements in image understanding tasks, the paradigm of object-oriented problem solving is naturally introduced in VISUAL. This paradigm focuses not only on objects, but also on the relationships among objects.

### 2.2. The principle of closeness

In an image, objects are more likely to have a close relationship if they are close in distance. For instances, two line segments in a local area may form a corner, or parallel lines, so they are more related to each other. On the contrary, two line segments are unlikely to be related if they are far apart, and if there are many other line segments between them.

## 3. Syntax of VISUAL

There are three parts in a VISUAL program: *object declaration, components relationships* and *object description.* We will use a working program as an example to explain the syntax of these parts. The function of this example is locate a line segment.

### 3.1. Object declaration

Object declaration is the first part of all VISUAL programs. Its syntax is as follows:

*declaration:*
  **object** *type-name identifier_list obj_information*
  **attribute** *type-name identifier_list*
  **var** *type-name identifier_list var_information*
*obj_information:*
  **export_to** ⟨*filename*⟩ **type** *type_specifier*
  **import_from** ⟨*filename*⟩
*var_information:*
  **related_to** ⟨*filename*⟩
  **of_type** {*type_specifier*}

*type-name* specifies the data type name of the identifier(s) in the *identifier_list*, and the *type_specifier* follows the rule of C programming language. In the following, **object**, **attribute**, and **var** are explained by examples.

(1) If the identifier are components of an object, their dada and corresponding map will be **import_from** *filename*. If the identifier is an object to be located, its data set and map will **export_to** *filename*. Examples are shown as follows:

**object** Pt(spt, ept) **import_from** ⟨*level_1*⟩
**object** Segment line **export_to** ⟨*level_2*⟩
**type** [typedef struct_Segment {
  Pt spt, ept; int theta; }
  Segment; ]

(2) If the identifiers are temporary objects in the program, it should be specified which data set and map those variables are related to or which data type they are. Examples are shown as follows:

**var** Pt(ept1, ept2, npt) **related_to** ⟨*level_1*⟩
**var** R *ray* **of_type** [typedef struct_R {
  Pt spt, ept; int theta; }
  R; ]

(3) If identifiers are attributes of the exported object, their values will be evaluated in the program. Two examples are shown as follows:

**ATTRIBUTE** int theta
lb12

The object declaration part could help users to organize a model-based vision system. The previous four examples show that a line segment is composed by edge points imported from ⟨*level_1*⟩. The data of line segment will be exported to ⟨*level_2*⟩. These line segments could compose polygons so that ⟨*level_2*⟩ will be imported by another VISUAL program which tries to locate polygons in ⟨*level_2*⟩, then export the data of polygons to ⟨*level_3*⟩. Therefore, through the object declarations, user can construct a hierarchical vision system. An example is shown in Fig. 1.

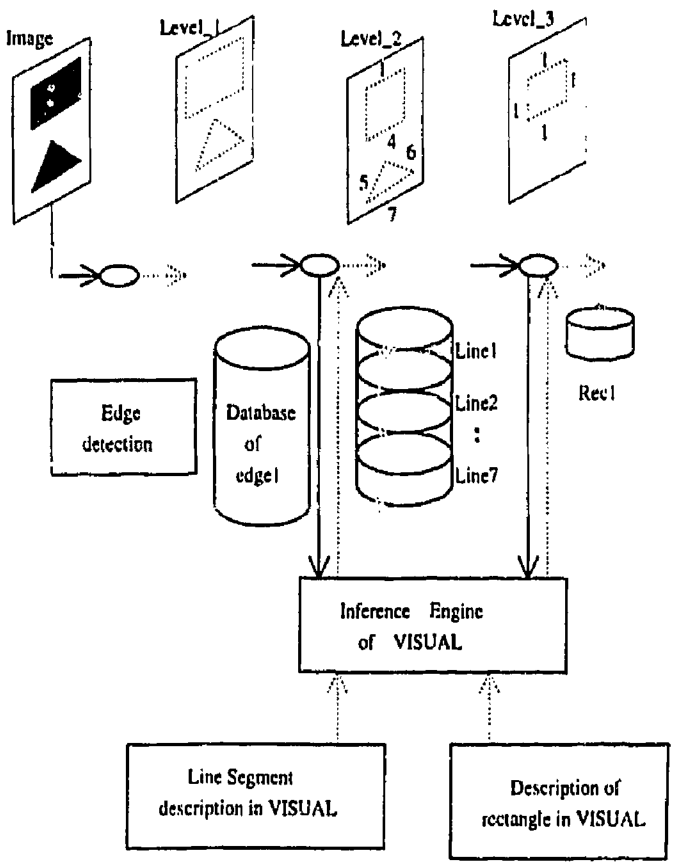
Image
Level_1
Level_2
Level_3
Edge detection
Database of edge1
Line1
Line2
:
Line7
Rec1
Inference Engine of VISUAL
Line Segment description in VISUAL
Description of rectangle in VISUAL

Fig. 1. A hierarchical vision system constructed by VISUAL. The diagram shows the input/output of the inference engine of VISUAL, and depict how to build a hierarchical model-based vision system.

### 3.2. Component relationship

An object is composed of its component objects, and its structure is determined by the relationships among the components. Users can define the relationship among the components in the second part of a VISUAL program. The syntax of this part is presented as follows:
*relation definition:*

**FUNCTION** *function-declarator function-body*

Both *function-declarator* and *function-body* follow the syntax of the *function-declarator* and *function-body* in C programming language. These functions, invoked by the inference engine of VISUAL, will test the relationship of the input parameters, and compute attributes of objects. An example is shown in Fig. 2.

### 3.3. Object description

The last part of a VISUAL program is the description of the exported object. It is enclosed by **begin** and **end**. Its syntax is presented as follows:

*description:*
  *obj-identifier* [*comp-and-attr-list*] → START (*identifier*): *expression*;
  var-identifier [*comp-and-attr-list*] → *expression*;
*expression:*
  *primary*

*primary. expression*
*expression* | *expression*
*primary:*
*fun-identifier* [ *parameter-list* ]
*var-identifier* [ *parameter-list* ]

A description begins with *obj-identifier* (an exported object) or *var-identifier* (a variable object). The var-identifier is mainly used to implement the technique of recursive call. The components and attributes of an object are explicitly specified in comp-and-attr-list. Every component of the comp-and-attr-list is prefixed by either ^ or _, which indicate whether the identity has been or is to be unified to an object in the database. If the identity is an attribute, these prefixes indicate whether the attribute has been computed or not.

The identifier in **start( )** represents a key component (or feature) of the object. Hansen and Henderson [10] enumerates the following five criteria for choosing key component: rareness, robustness, completeness consistency, and cost. Actually, choosing a proper key component can greatly reduce total number of matching trials. For example, Koenuka and Kanade [11] choose angles and parallel lines as key features to match internal models against 2-D shape data, and reduce the total match trials by the factor of $10^9 - 10^{11}$. Each expression is group of object descriptions. Each group is separated from the others by "|". *Primary* in a group is separated by ",". In Fig. 3, an example demostrates every control structure in the object description part of VISUAL. The function of the example is to locate line segment. The object *line* is described by two ending points, spt and ept, and theta, which represents the direction line. The key component of *line* is spt which is a starting point of *line*. The property of spt is verified by the function StartPoint( ). Ray is a *var-identifier*. The value of spt in ray's *comp-and-attr-list* is known, but ept is not. The description of ray is a recursive call or the function EndPoint( ), which means the ray reaches its end. Thus, by the description of line, all the possible line in ⟨*level _ 1*⟩. Will be automatically located by the inference engine of VISUAL.

```
function Next (ept1, ept2, theta)
  Pt *ept1, *ept2;
  int * theta;
  {/* test whether point ept2 is next to ept1 */
  int dx, dy, d_ th;
  double len;
  dy = (ept1 → y − ept2 → y);
  dx = (ept1 → x − ept2 → x);
  d_ th = abs(ept1 → theta − ept2 → theta);
  len = sqrt (double) (dx * dx + dy * dy));
  if ((len < = 3.0) & & (d_ th < 2))
     {chi → degree + = ept2 → degree;
     return(1); }
  else return(0); }
```

Fig. 2. An example of the component relation part in the VISUAL problem.

```
begin
   line [_ spt, _ ept, theta] →
start (_ spt):
   Start Point (^ spt),
   ray[^ spt, _ ept, theta];
   ray[^ ept1, _ ept2, ^ theta] → df
   Next(^ ept1, _ npt, ^ theta),
   ray[^ npt, _ ept2, ^ theta] |
   End point (^ ept1);
end
```

Fig. 3. An example of the description part in VISUAL program. The function of the example is to locate line segment.

## 4. Implementation of VISUAL

Since VISUAL is an object-oriented and declarative language, there is an inference engine (INFG) in VISUAL to perform unification. The input of INFG is an object description written in VISUAL. The outputs of INFG are a database and a map. There are two major components and four precedures in INFG. The two major components are internal representation and the chromatographic search(CSH). The four procedures are initialization, pre-processing, unification, and plotting.

## *4.1. Internal representation*

The internal representation of an object is based on the principle of object orientation. The object type is defined in the declaration part of a VISUAL program. Objects of the same type are stored in the same database and the boundary points of there objects are depicted in a map. These points in the map are numbered according to the indices of corresponding object records in the database. For example, a rectangle is composed by line segments, $l_1$, $l_2$, $l_3$, $l_4$ which records are stored in the position 1, 2, 3, 4 of the database, repectively, so the edge points of $l_1$ are all numbered 1; $l_2$, 2, and so on. Therefore, this numbering becomes the linkage between the object in the map and its corresponding database. In addition, this numbering realizes the CSH.

## *4.2. Chromatographic search*

The CSH is the realization of principle of object orientation and principle of closeness. Because the boundaries of objects in the map are numbered according to their record indices in the databass, CSH directly searches nearest objects on the map after given an object. The process of CSH is like a radar which spreads out signals in all directions from a given point. When signals encounter an object, the number of the object is reported. The CSH has the following two merits:

(1) *Polymorphism.* The CSH is polymorphic because it can find objects of any types. The polymorphism of CSH is realized by maps. Although different objects in databasses have different data types, the type of their corresponding maps is integer array. Therefore, CSH works on 2-D maps of homogeneous type rather than on databasses of different types.

(2) *Omnidirection.* In a database, data items are arranged in a certain order. However, Searching a key which does not follow as certain order may take O($n$) comparisons, if the database has $n$ records. Otherwise, it takes O($n$ log $n$) to sort these items again. Based on the principle of closeness, CSH searches most related objects in all directions on the map, therefore the arrangment order of the data items in the database is not important.

The time complexity of CSH on an $N \times N$ map is O($N^2$) of it is executed on s serial computer. However, it is only O($N$) if on a mesh-connected computer.

## *4.3. Four procedures*

There are four procedures in INFG. They are initializeation, pre-processing, unification and plotting. Their functions are describe as follows.

(1) *Initialization:* The first step of INFG is initialization. The files regarding to imported components and the exported object are opened, then the database and maps components are installed.
(2) *Pre-processing:* The relationships among components are extracted from the VISUAL program. The control structure, **and**, **or**, or recursive call in the VISUAL program, are translated to C codes for unification.
(3) *Unification:* Unification is a process which determines the values for variables. This procedure takes indexes provided by CSH, and access the data in the database. It will assign at most k values to a variable using the principle of closeness, therefore, the computation time is reduced.
(4) Plotting procedure: The output of the INFG include a map, so there needs a procedure to draw the boundary of objects. If some components satisfy all of the pre-defined relationships, then one of the described object is found. The boundary points of the object are determined by the boundary points of the satisfactory components.

INFG applies the technique of dynamic programming to locate the described object, and use the principle of closeness as a pruning technique, so the time complexity of INFG depends on CSH, the plotting procedure and the number of the key component. If an object is composed by $m$ component and a key component which has $n$ instants in the corresponding database, INFG will instantiate $n$ hy-

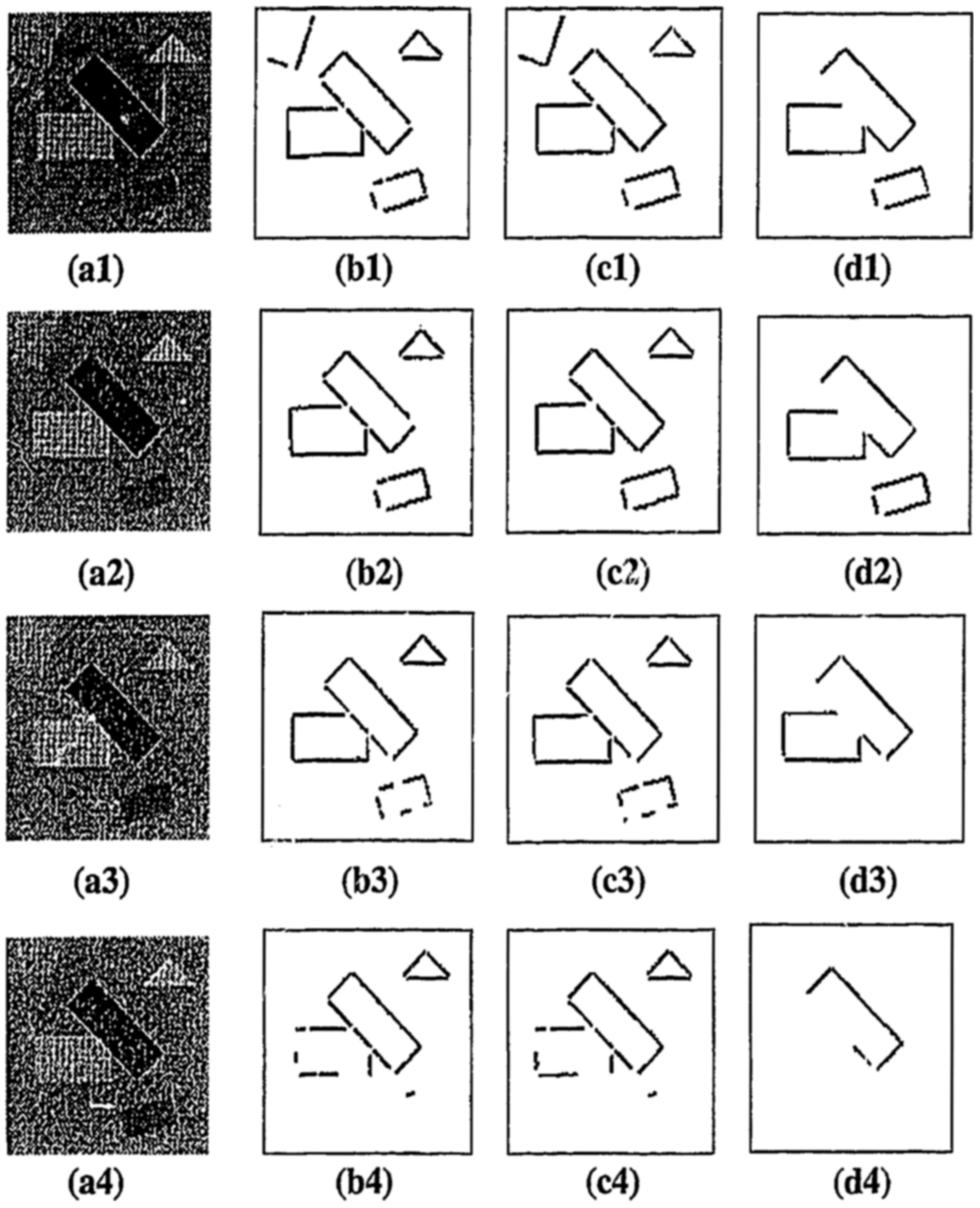

Fig. 4. A sequence of synthesized images with different levels of noise. (a$i$) is original image with Gaussian noise of standard deviation $2^i$ where $i = 1, 2, 3, 4$. (b$i$), (c$i$) and (d$i$) are results of edge detection, locating line segment, and locating rectangle in (a$i$), respectively.

potheses to be verified. Each unification of a component calls CSH a time, hence, the time complexity of locating the object in an $N \times N$ image is $O(n^k \times m^k \times N)$, where $k$ is the number of nearest neighbors found by CSH pruning.

## 5. Experiments

In this section, we demonstrate the power of VISUAL programs on image understanding. These sample images are classified into two sets: a set of synthesized images, and a set of real images with a stop sign. We use images of the first set to test the performance of two VISUAL programs under different noisy envirements. These programs will locate line segments and rectangles, respectively. Images in the second set are extracted from street scenes. Each of these images contains a stop sign which is an important landmark for robotic navigation. At the end of this section, we will show the possible ambiguity while describing objects.

### 5.1. Synthesized image set

Figs. 4(a$i$), for $i = 1$ to 4, are synthesized images with Gaussian noise of standard deviation $2^i$, and

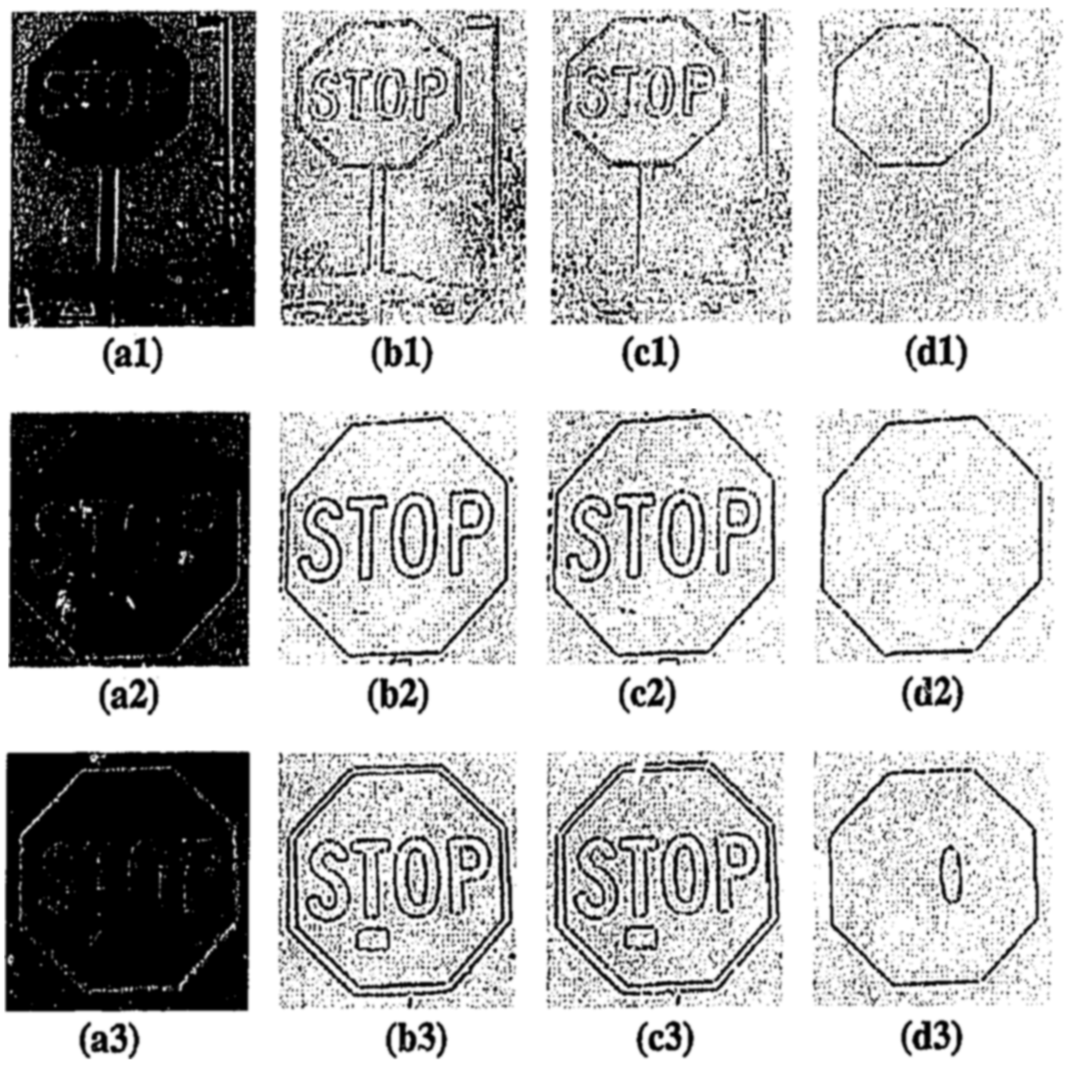

Fig. 5. (a1), (a2) and (a3) are real images with stop signs under different illumination. (b$i$), (c$i$) and (d$i$) are results of edge detection, locating line segment, and locating octagon, respectively.

Figs. 4(b$i$) are the results of $X^2$ feature detector with threshold of crirical region of size 0.10 [12]. If we apply the example VISUAL program shown in the previous section, and apply the Mahalanobis distance for the principle of closeness, broken line segments in (b$i$) are connected, as in Fig. 4(ci). The Mahalanibis distance which is also called the $X^2$ statistic is the fundation of the ecidence combination theory [12]. It can improve the perforemence of Hough Transform which traditionally use edge points as evidence, instead of $X^2$ statistic generated directly from hypothesis tests.

The VISUAL program in the Appendix can locate rectangles which components are line segments. If a rectangle is defined as four line segments which are prependicular and connect to their neighbor except the last one, the results are shown in Figs. 4(d$i$). This definition demonstrates the importance of the relationships among objects. Unfortunately, ambiguity also originates from the definition. We will discuss this topic later in Section 5.3.

### 5.2. Stop sign set

Images in this set contain stop signs which are important landmarks for robotic navigation. These images are digitized from pictures taken from a street scene under different illumination. For examples, these pictures are taken in the morning, afternoon (facing the sun), and at night. We wrote a program in VISUAL to locate stop sign candidates in a street scene. Because octagons are rate in the natural world, we treat the octagons as strong candidates of stop signs.

An octagon is a polygon composed of eight line segments, and each angle is 135 degree angles. We encoded this definition into a VISUAL program. Figs. 5(a2) and 5(a3) are extracted from the same street scene as Fig. 5(a1), but different illuminations so that the intensities or their backgrounds are different. For example, we were facing the sun while taking picture in the afternoon. The background of the stop sign is too bright that the out-ward boundary lines of the stop sign are hard to be detected. The outward boundary lines are broken in most cases, but the inward ones still remains detectable.

Fig. 5(d3) is an interesting result. Two octagons are found in the image. Actually the small one is an ellipse. The ellipse turns out to be an octagon after the process of digitization and locating line segments. This result also shows that if the definition for a stop sign candidate is an equilateral octagon, then the rate of false position will be decreased.

### 5.3. Ambiguity in object descriptions

If a user does not define an object clearly and precisely, the IFNG of VISUAL may locate wrong objects. For example, if we define a triangle as three line segments which connect each other, then the

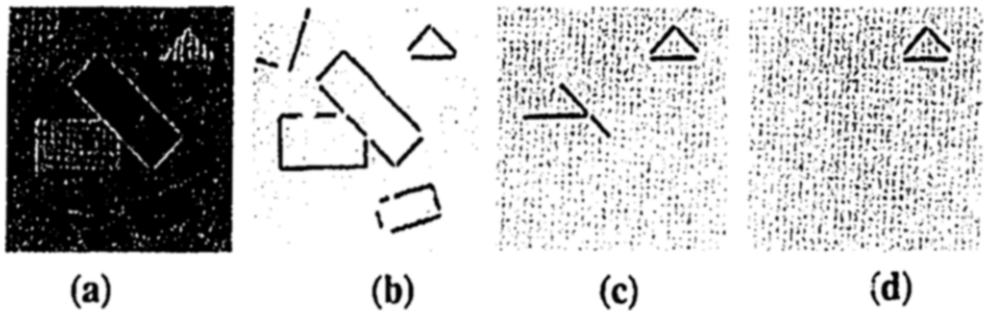

Fig. 6. Ambiguity in object descriptions. (a) is the original image, (b) is the result of locating line segment. (c) and (d) are the results of two VISUAL programs with different descriptions of triangle.

INFG gets two "triangles" from Fig. 6(a), as shown in Fig. 6(c). These "triangles" show the problem with this definition of triangle. After including the condition that sum of internal angles is 180 degrees, the real triangle is found, shown is Fig. 6(d). A deep question is how to define an object precisely. This question is related to cognition science, and is beyond the scope of this paper.

## 6. Conclusion

In this paper, we present on object-oriented and declarative language, VISUAL, for image understanding task. The design of VISUAL is based on the *principle of closeness* and *principle of object orientation*. It is realized by the chromatographic search on 2-D component maps, instead of searching on database for unification. The inference engine of VISUAL will open corresponding databases and images according to given description, then perform unification. Hence, VISUAL is the integration of programming language, database, and image. It is also a tool to build a hierarchical vision system and combination of homogeneous evidence.

## Appendix. A VISUAL program for locating rectangles

```
object Segment(11, 12, 13, 14) import_from
    ⟨level_2⟩
object Rec rectangle export_to ⟨level_3⟩
type [ typedef struct_Rec { Segment s1, s2, s3, s4;
  int degree; double statistic; } Rec; ]
begin
  rectangle[_11, _12, _13, _14] →
start(_11):
  Perpendicular(^11, _12),
  Perpendicular(^12, _13),
  Perpendicular(^13, _14),
  Perpendicular(^14, _11);
end
```

## References

[1] C.-H. Chen and P.G. Mulgaonkar, Automatic vision programming, *CVGIP-Image Understanding* 55 (1992).

[2] R.O. Duda and P.E. Hart, *Pattern Classification and Scene Analysis* (John Wiley and Sons, New York, 1973).

[3] H.S. Don and K.S. Fu, A syntactic method for image segmentation and object recognition, *Pattern Recognition* 18 (1985).

[4] K.S. Fu, *Syntactic Pattern Recognition* (Prentice-Hall, Englewood Cliffs, NJ, 1982).

[5] B. Bhanu, Cad-based robot vision, *IEEE Computer* 20 (1987).

[6] P. Flynn and A. Jain, Bonsai: 3-d object recognition using constrained search, *PAMI* 13 (1991).

[7] D.M. McKeown, W. Harvey and L. Wixson, Automating knowledge acquisition for aerial image interpretation, *CVGIP* 46 (1989).

[8] B.G. Batchelor, *Intelligent Image Processing in Prolog* (Springer, London, 1991).

[9] K. Fuchi, Launching the new era, in: *Proc. Internat. Conf. Fifth Generation Computer System*, ICOT, 1992.

[10] C. Hansen and T. Henderson, Toward the automatic generation of recognition strategies, in: *Proc. 2nd Internat. Conf. on Computer Vision, IEEE* (1988) 275–279.

[11] Koezuka and T. Kanade, A technique of pre-compiling relationship between lines for 3d object recognition, in: *Proc. IEEE Internat. Workshop on Industrial Application of Machine Vision and Machine Intelligent* (1987) 144–149.

[12] E.R. Chuang and D.B. Sher, X2 test for feature detection, *Pattern Recognition* (1992).