



University of Pennsylvania
ScholarlyCommons

Technical Reports (CIS)

Department of Computer & Information Science

August 1984

The Recognition and Representation of 3D Images for a Natural Language Driven Scene Analyzer

Amy Zwarico
University of Pennsylvania

Follow this and additional works at: https://repository.upenn.edu/cis_reports

Recommended Citation

Amy Zwarico, "The Recognition and Representation of 3D Images for a Natural Language Driven Scene Analyzer", . August 1984.

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-84-29.

This paper is posted at ScholarlyCommons. https://repository.upenn.edu/cis_reports/832
For more information, please contact repository@pobox.upenn.edu.

The Recognition and Representation of 3D Images for a Natural Language Driven Scene Analyzer

Abstract

Two necessary components of any image understanding system are an object recognizer and a symbolic scene representation. The LandScan system currently being designed is a query driven scene analyzer in which the user's natural language queries will focus the analysis to pertinent regions of the scene. This is different than many image understanding systems which present a symbolic description of the entire scene regardless of what portions of that picture are actually of interest. In order to facilitate such a focusing strategy, the high level analysis which includes reasoning and recognition must proceed using a topdown flow of control, and the representation must reflect the current sector of interest. This thesis proposes the design for 3 goal-oriented object recognizer and a dynamic scene representation for Landscan - a system to analyze aerial photographs of urban scenes. The recognizer is an ATN in which the grammar describes sequences of primitives which define objects and the interpreter generates these sets of primitives. The scene model is dynamically built as objects are recognized. The scene model represents both the objects in the image and primitive spatial relations between these objects.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-84-29.

**THE RECOGNITION AND
REPRESENTATION OF 3D IMAGES
FOR A NATURAL LANGUAGE DRIVEN
SCENE ANALYZER**

**Amy Zwarico
MS-CIS-84-29
GRASP LAB 20**

**Department Of Computer and Information Science
Moore School
University of Pennsylvania
Philadelphia, PA 19104**

August 1984

Acknowledgements: This research was supported in part by DARPA grants NOOO14-85-K-0018 and NOOO14-85-K-0807, NSF grants MCS8219196-CER, MCS-82-07294, 1 RO1-HL-29985-01, U.S. Army grants DAA6-29-84-K-0061, DAAB07-84-K-F077, U.S. Air Force grant 82-NM-299, AI Center grants NSF-MCS-83-05221, U.S. Army Research office grant ARO-DAA29-84-9-0027, Lord Corporation, RCA and Digital Equipment Corporation.

MS-CIS-84-29

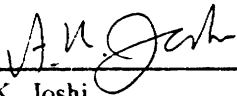
UNIVERSITY OF PENNSYLVANIA
THE MOORE SCHOOL OF ELECTRICAL ENGINEERING
SCHOOL OF ENGINEERING AND APPLIED SCIENCE

THE RECOGNITION AND REPRESENTATION
OF 3D IMAGES FOR
A NATURAL LANGUAGE DRIVEN
SCENE ANALYZER

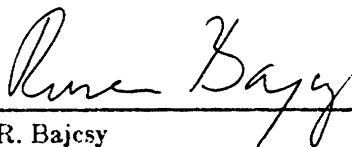
Amy Elizabeth Zwarico

Philadelphia, Pennsylvania
August 1984

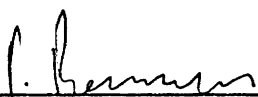
A thesis presented to the Faculty of Engineering and Applied Science
of the University of Pennsylvania in partial fulfillment of the
requirements for the degree of Master of Science in Engineering for
graduate work in Computer and Information Science.



A.K. Joshi



R. Bajcsy



Graduate Group Chairperson

ABSTRACT

Two necessary components of any image understanding system are an object recognizer and a symbolic scene representation. The LandScan system currently being designed is a query driven scene analyzer in which the user's natural language queries will focus the analysis to pertinent regions of the scene. This is different than many image understanding systems which present a symbolic description of the entire scene regardless of what portions of that picture are actually of interest. In order to facilitate such a focussing strategy, the high level analysis which includes reasoning and recognition must proceed using a top-down flow of control, and the representation must reflect the current sector of interest. This thesis proposes the design for a goal-oriented object recognizer and a dynamic scene representation for LandScan - a system to analyze aerial photographs of urban scenes. The recognizer is an ATN in which the grammar describes sequences of primitives which define objects and the interpreter generates these sets of primitives. The scene model is dynamically built as objects are recognized. The scene model represents both the objects in the image and primitive spatial relations between these objects.

ACKNOWLEDGEMENTS

I would like to thank my advisors Dr. Joshi and Dr. Bajcsy for their help and continuing support.

Special thanks goes to Tim Finin for his help with the design of the ATN interpreter and Isaac Miller for his help with Scribe.

I am grateful to Jon Korein and Gerry Radack for the time they spent explaining some of the intricacies of SurfsUP. Without them, the scene model would have been a dream; its implementation a nightmare.

Eric Krotkov deserves the credit for the implementation of the surface model.

I would also like to thank Sharon Stansfield who has listened to a lot of complaining and random ideas throughout the course of this work.

Finally a special thank you to my family for their constant support and patience.

This work was supported by U.S. Airforce grant F49620-83-K0037.

Table of Contents

1. Introduction	1
2. Background	3
2.1. LandScan	3
2.2. The ATN Formalism	10
2.3. SurfsUP	16
3. Object Recognition	19
3.1. Justification for the Use of an ATN	20
3.2. The Object Recognition Grammar	25
3.3. The Dictionary	28
3.4. The ATN Interpreter	30
4. Representation - The Scene Model	34
4.1. The Basic Design Criterion	35
4.2. The Object List	37
4.3. The Relations	38
4.4. Justification of the Scene Model	40
4.5. Implementation of the Scene Model	41
5. Future Work	44
6. Conclusion	47
APPENDIX A. ATN Grammar for Object Recognition	50
APPENDIX B. Scene Model Data Structures	57

List of Figures

Figure 3-1: A Sample Grammar	25
Figure 3-2: Major Types and Subtypes of Objects	27

CHAPTER I

Introduction

In image understanding an image is given to a computer and is processed to produce a symbolic description of the image. Image understanding can be extended by making the analysis goal-oriented. Goal-oriented scene analysis restricts the analysis to those areas of the scene which are of current interest. This requires there to be some interface which will enable the user to indicate just what portions of the image are of interest to him/her. The interface can be a symbolic computer language though this restricts the use of the system to those people who know the language. On the other hand, a system which uses natural language to drive the image understanding process will be accessible to all potential users. LandScan (Language Driven Scene Analyzer) is being designed to perform natural language query driven scene analysis of 3D images of aerial views of urban scenes.

In order to do any reasoning about a scene it is necessary to both interpret and represent the image data using some symbolic description. A further constraint on LandScan is that it must construct a representation of the scene which will facilitate the interface to the natural language driver. This means that not only must the actual scene be represented in a way which enables high level reasoning to be performed, but global knowledge necessary for the interpretation and generation of natural language utterances must be encoded. The representation must also be dynamic; constructed as the system is queried. Otherwise the natural language interface will be nothing more than a simple question/answer system for a knowledge base which was previously constructed using no special focussing mechanisms.

LandScan will ultimately reason about urban scenes using natural language queries to guide this analysis. In order to accomplish this it must recognize objects in a scene and assign labels to them. It must also store the information about the scene - the objects in it and the topological relations between the objects. The model of the scene - objects and the relations between objects - must then be represented in some manner which will facilitate the following reasoning operations to be performed on the scene data:

1. determining the existence of an object
2. finding an object part
3. determining locative relations, both simple and complex, among objects.

In order to interpret the natural language queries driving the high level analysis processes the system must also have available to it global knowledge which includes the linguistic features of the objects in the domain. These linguistic features will enable the system to correctly interpret natural language utterances.

This thesis will propose a solution to the problem of the recognition and representation of 3D objects in aerial views of urban scenes to be used by a language driven scene analyzer. An Augmented Transition Network (ATN) has been chosen to perform the object recognition because it has a top-down flow of control thus facilitating the interface between the queries and the recognition process. The scene will be represented symbolically by the objects which have been recognized and the primitive spatial relations which hold between them.

An overview of LandScan, and discussions of SurfSUP (the modelling system chosen for implementation of the scene model) and the ATN formalism will be presented first. Next the design of the ATN used for object recognition will be discussed. Following this the scene model will be described. Finally, a discussion of the work remaining to complete the LandScan system and some future extensions which could be made to LandScan will be presented.

CHAPTER II

Background

This chapter presents an overview of LandScan, a description of the general ATN formalism, and a brief discussion of SurfsUP - the modelling system chosen for implementing both the surface and scene models.

2.1. LandScan

Given a pair of stereo images, LandScan will be able to correctly answer natural language queries about the scene by using low and high level vision and reasoning processes. In the following section the various components of the analyzer will be described.

LandScan has four major components which transform the data from a digitized image into answers to natural language queries about the scene. The low level image processing routines detect edge points (edgels), perform stereo matching to obtain the 3D information crucial to the higher level analysis, and segment the picture into various picture primitives - edges and regions. The middle level imaging modules add topological properties to the regions detected by the low level routines so that these regions may be grouped together in the high level image representation. It is the high level vision which uses the image primitives (regions), the topological properties of this set of regions, and *a priori* knowledge of the domain to recognize that a subset of these regions is in fact a single object and to assign a label to this set of faces. The high level image processor also determines and stores a very primitive set of spatial relations which hold between the objects it has identified. Finally, the natural language interface and high

level reasoner parse queries, search the representation of the image for the data in question, and generate the appropriate response to the query.

All low and middle level image processing is performed in a bottom-up fashion when the digitized image is presented to the system for analysis. The high level vision analyzes the scene using a top-down control structure driven by the queries parsed by the natural language interface. No high level recognition or analysis is performed until a query is made. When a question is asked, the high and middle level models of the image are analyzed only as much as is necessary to enable the generation of an adequate response for the user. Since the high level analysis of the scene is driven by the queries of the user, the high level vision processes operate in a top-down fashion. Only those objects and relations expressly mentioned in the query are searched for in the image. Thus it is not always the case that a scene will be completely analyzed. Only those portions of the scene which are of interest to the user will be analyzed. Although using this recognition strategy will increase the time necessary to answer a question, it will probably reduce the overall amount of work which is done in analyzing a scene. The system will only store a representation of the parts of the scene which have been asked about.

The polyhedra was chosen as the image primitive for the representation of objects. This choice was not arbitrary, but carefully considered in the system domain, aerial photographs of urban scenes. Looking at such images they appear to be composed of polyhedra of various sizes and shapes at different distances from the ground. Therefore it is very intuitive to describe the representation of objects in terms of sets of these polyhedra. The final symbolic representation of the scene will be labelled sets of polyhedra. The decision to model the scene by collections of planar polyhedra meant that the planar representation of a scene would have to be implemented. Available was SurfsUP which contains almost all the data types and routines necessary to do such modelling. Where SurfsUP was lacking, it provided the flexibility necessary to extend the system to accommodate the model.

The system being designed has many different processes which work on various representations of the data available from the scene to make the final analysis possible. Each of these operations needs a different representation of the data - pixels, faces, objects, etc. The scene is represented by a hierarchical model with entities at level i being pointers to collections of entities at level $i-1$. The various levels of representation in the hierarchy are pixel, edgel*, stereo matched edgel, segment, region, surface, and scene. The pixel representation is an $N \times N$ matrix of grey levels (between 0 and 255) representing the "color" of that point in the digitized image. The edgel model represents all of the edge points (whether real or noise) which have been found after convolving the image. Again, this is an $N \times N$ matrix where a zero indicates no edge point and a non-zero value is the change in value of the two regions which the edge separates. The system then uses the stereo pair of edgel models to obtain a representation of the 3D information about the regions bounded by the edges. Again, an $N \times N$ matrix is used for this representation. The edge points are then approximated by segments and the representation produces a list of line segments determined by their end points. These segments are combined into polyhedra: each polyhedron is defined by a list of edges defining it, its surface normal, shape - the number of edges defining it, its compactness and its centroid. This surface model also has the relations between polyhedra stored. These relations are necessary for performing recognition tasks. The highest level in the hierarchy is the scene model. It consists of a list of objects in the scene and the relations between those objects. Each object is described by the set of faces composing it and the specific attributes of this object - its location and label. The relations express the topology of the scene - in a very primitive way the location of objects relative to one another.

In order to perform the high level vision and reasoning tasks required by this system, world knowledge must also be encoded into the system. This global knowledge will be used to recognize

* an edgel is a pixel corresponding to a boundary in the image

objects, understand natural language queries, and search the scene model to obtain an answer to a query. Presently, three sources of *a priori* knowledge have been determined necessary to perform the above mentioned tasks. These three representations of world knowledge are an ATN grammar which describes the manner in which surfaces are grouped to form objects, a world model and an object model. The world and object models are very similar to those in the Shapiro and Haralick [Shapiro 84] system as well as Rosenthal's Conceptual Hierarchy [Rosenthal 81].

The Grammar rules used in recognizing objects have been structured as combinations of surfaces and objects because of the appearance of these urban objects in an aerial view. A sequence of rowhouses appears to be a sequence of simple buildings joined together in a particular fashion. A divided highway looks like several roads aligned in a specific manner. The grammar presents a straightforward description of the steps which must be followed in order to construct a specified object type from the data primitives present in the actual scene. It is also a good representation scheme for a domain in which objects of the same type (ie. building) may have very diverse appearances.

Unlike Rosenthal's Conceptual Hierarchy for modelling world knowledge, the world knowledge has been divided into two levels - the world model and the object model. Rosenthal proposed a purely hierarchical model of the world which is ordered by two ON relations - ON TOP OF and PART OF. ON in his system is essentially a size ordering. What this implies is that the entities in the Conceptual Hierarchy are not at the same level of abstraction. Therefore, CAR ON STREET and VENT ON BUILDING, while both are represented by the Conceptual Hierarchy, mean very different things. This makes the Conceptual Hierarchy an unwise choice of representation of global knowledge for a system which must reason based on natural language queries. Instead a representation similar to Shapiro and Haralick's hierarchical, relational model was chosen [Shapiro 84].

The problem faced by Shapiro and Haralick is analogous to the problem at hand - design a system which can perform a variety of tasks on the same data - and their solution was very influential in the design of the image model. Shapiro and Haralick have designed a hierarchical, relational model of a domain (an F-15 bulkhead) which provides precise, accurate information for low-level vision and at the same time is useful to high-level vision and planning processes. Their model has four levels in the description hierarchy of the scene - the world, object, part and surface/arc divisions. Each level has a spatial data structure consisting of a set of relations giving the features and relationships of the entities at that level. The world level models the arrangement of objects in the world. The object level - the arrangement of parts in each object. The parts level describes the relationships between surfaces of a part. The surface/arc level specifies the actual surfaces and arcs at a physical point level. With such a data structure all the desired operations can be performed.

Like the hierarchical relational model of Shapiro and Haralick, the world model describes the features and relations of the objects in the domain. The objects are those which can be expected in an urban scene - buildings, streets, sidewalks, etc. The world model represents the relations between objects in the domain. The world is represented by a labelled directed multigraph in which the nodes are the objects in the domain and the arcs are labelled with the relation which holds between the two objects in the world. It has been determined that at least two relations are needed to adequately model the world. The two relationships are NEXT_TO and ON. NEXT_TO implies that two objects can be expected to be adjacent in the domain. This adjacency does not necessarily mean that the two objects will be adjacent in the geometric sense - sharing a common boundary - but that they would be viewed as being "close enough" to be considered adjacent. For example, CAR NEXT_TO BUILDING could be said to hold even if the car and building are separated by some other small object such as a sidewalk. The ON relation has one interpretation "on top of". In the current domain CAR ON STREET is a valid relation but BUILDING ON

STREET is not. We do not expect to see buildings on top of streets in urban scenes. CAR ON STREET can only mean that the object CAR can be on top of an object STREET, not that the CAR can be part of the STREET. The information about the properties and features of the objects is represented in the object model. This way, there is no possible confusion about the meaning of the relationships represented by the arcs in the graph.

The object model represents the expected physical features and linguistic properties of the objects in the domain. The physical properties are the parts of objects. These "parts" are the objects which were not included in the world model in order to keep the level of abstraction in that model consistent. In the object model, objects are decomposed into their possible parts - a BUILDING can have the parts roof, walls, steps, vents, and airconditioners just to mention a few. Since the objects in the world model are not prototypes per say, there will simply be a list of parts associated with each object. There will be no indication of the number of each part which the object should have.

The linguistic properties are those features which affect the usage and interpretation of a spatial construct (phrases describing the spatial relations between objects). Since the domain is a visual one, each object in the domain will have a "place" associated with it. However, since objects have different qualities associated with them, they may occupy space - have a place - in different ways. This notion of place is what Herskovits calls the canonical geometric description of a spatial entity (what I am calling objects) [Herskovits 82]. Most of the objects in the domain will be ordinary solid objects which are bounded, closed surfaces. These are objects such as buildings, vehicles and people. The other major category of objects in this domain are the geographical objects. Geographical objects are entities with slightly imprecise boundaries - roads, rivers, and fields. Although most of the entities in the domain fall into one of these two categories, there are several other descriptions which will have to be considered in order to design

a robust natural language interface to the system. These are the geometric descriptions for liquid or gaseous objects, parts, a group of entities considered to occupy a single place, geometric objects such as points of lines, parts of space, holes, unbounded objects, and substances.

The notion of the space which an object or entity occupies is only one of the types of object knowledge which is necessary for proper interpretation and analysis of spatial constructions. Each object must have particular knowledge associated with it. Some of these properties are a prototype shape and the allowable deviations from it, the relative size, and characteristic orientation - ie. a table stands on its legs normally. The typical geometric conceptualization will also affect the choice of spatial construct - is the object normally viewed as a point or line. Along with the typical geometric conceptualization is the typical physical context of an object. For instance, a door is normally viewed as being in a wall. The normal function of an object, its functionally salient parts and the actions commonly performed with an object will also be necessary for analyzing the spatial constructs.

To reiterate, LandScan is a query driven scene analyzer for 3D aerial views of urban settings which uses low, middle and high level vision processes, high level reasoning and natural language understanding to analyze an image. The low and middle level vision processes are data driven and are performed when LandScan is given the image input. These processes include edge detection, stereo matching, various segmentation processes and the construction of the surface model. The high level image processes which are driven by the queries are the subject of this thesis. They are top-down object recognition using the ATN formalism, and the construction of the scene model. The reasoning processes are goal oriented searching for an adequate answer to the query which is processed by the natural language interface. The high level vision, reasoning and language understanding processes use global knowledge which is represented for LandScan in three models: the ATN grammar for object recognition, the world model and the scene model.

Using all of these components LandScan will analyze those parts of the scene which are of interest to the user.

2.2. The ATN Formalism

Augmented Transition Network grammars (ATNs) have been used in natural language understanding systems (both written and spoken) for at least ten years. They have proved useful because of their perspicuity, generative power, efficiency of representation, their ability to capture linguistic regularities and generalities, and efficiency of operation. They are also easy to interface with other components of a system. In general, the ATN formalism has been used as a top-down, left-to-right parsing system for natural language sequences.

The following section will describe the ATN formalism as appears in Bates [Bates 81], Winograd [Winograd 83] and Winston [Winston 79] [Winston and Horn 81], and the general structure of an ATN interpreter (there are several such structures). For additional information the reader is referred to the above references as well as the original description of the formalism as found in Woods.

A basic transition network or finite automata is a set of states connected by arcs. Finite automata have long been used in formal language theory as convenient ways of encoding a set of sequential patterns for syntactic recognition or generation. A finite automata consists of a set of states and a set of transitions (or arcs) from state to state that occur on input symbols chosen from a designated alphabet. Each automata has a state which is designated as an initial state and a set of final states. The states of the finite automata correspond to the nodes of the transition network while the transitions correspond to labels on the arcs of the network. A network is said to accept an input sequence if there is some transition from the designated starting state to some final state. For further discussion of finite automata and transition networks the reader is referred to Hopcroft [Hopcroft and Ullman 79].

A simple finite automata is only equivalent to a regular expression and it can be easily shown that regular expressions are not adequate for modelling many "languages". The Recursive Transition Network (RTN) is weakly equivalent to a context free grammar or a push down automata. An RTN appears to be a collection of finite state transition networks - a set of directed graphs with labelled states and arcs, and distinguished start and final states. It differs from a simple transition network in that it permits recursion by allowing the labels on the arcs to be both terminal symbols from the language being described and the name of another transition network (a non-terminal symbol of the grammar associated with the network).

An ATN is an augmented RTN. It consists of a set of states, arcs which are classified by the type of transitions they represent, and a set of registers representing features and roles of the constituents of an input string. The arcs also have conditions and actions associated with them. If the set of conditions on an arc hold for the current input and register symbol, the arc is traversed and the associated actions are carried out causing the registers as well as the state to be changed. This augmentation of the structure gives an ATN the power of a Turing machine. This is so because while the ordering of the states and arcs reflects the surface structure of input sequences of the language being modelled, the actions allow rearrangements of the input sequence to represent a "deep structure" of the input sequence.

The states in an ATN correspond to particular points in the grammar being reached. The arrangement of these states reflects the acceptable surface structure of input sequences. In following Bates' description of the formalism, this is reflected in the choice of state names. In general, a state name will be composed of two parts. The first part of the name indicates the constituent being processed, while the second part indicates either how far through the constituent the parse has proceeded or the construction which might occur next. Mnemonic names are not necessary (Winograd does not use them) but they help to clarify the grammar for human users.

The registers in an ATN are like variables in a program - each having a name and a particular type of information which can be stored in it. These registers basically represent the properties of the network with which they are associated. The registers are of two types: feature or role registers [Winograd 83]. The feature registers can hold values chosen from a finite set of possibilities. For example if there were a NUMBER register associated with a network it might hold the value SINGULAR or PLURAL. The role register can hold a node or a sequence of nodes representing a constituent which has been found. The nodes in a role register are parts of the structure which have already been constructed by the network. The role registers are used to capture the fact that two ostensibly different sequences have the same underlying structure. There is also a third type of register corresponding to a global variable. This is the HOLD register which is accessible to all graphs in the ATN. It is used to store constituents which have been constructed by the ATN but as yet have no "role" in the final structure. The contents of the HOLD register are used when a constituent is needed but does not appear at that point in the input sequence. Unlike the other two types of registers, the contents of the HOLD register are deleted when another register is set from the HOLD register.

The arcs of an ATN are no longer simply labelled by a terminal symbol from the language defined or a non-terminal naming some other network in the set of graphs comprising the transition network. The arcs in an ATN are divided into seven different categories [Bates 81]. The interpretation of an arc in an ATN depends upon the type of arc it is. The CAT arc tests to see whether the current input word is of the specified category. A WRD arc is even more specific, testing whether the current input and word specified on the arc are in fact the same. A MEM arc allows the current input to be a member of the list of words associated with a particular arc. These three types of arcs all consume a single word in the input sequence as they are traversed. A JUMP arc corresponds to the empty-transition of formal language theory. It specifies that the arc is to be traversed without consuming any input. The VIR arc tests to see whether a

constituent of the named type has been placed on the HOLD list. The PUSH arc (called a SEEK arc in [Winograd 83]) initiates a call to another network in the ATN to look for a particular constituent. The POP arc (SEND in [Winograd 83]) has no destination state. It marks the state it leaves as a final state of that subnetwork. It indicates the form or structure of the input data which is to be returned to the state which called that subnetwork. If the network was invoked by a PUSH arc, control is returned to the PUSH arc which called the network when a POP is encountered.

As mentioned above, actions construct pieces of structures and store them in the registers associated with that level of the ATN. There are three basic types of actions (although some systems have added other kinds of actions which have been found necessary for the particular application). The three basic actions are SET, SEND, and LIFT. SET assigns the specified value to the register. If it is a feature register this value is a feature dimension choice, for a role register, the value is a node. Role registers can either be set by a completely new node or have a node appended to the current contents. The other two types of actions are used to approximate parameter passing. The SEND action appears only on PUSH arcs. It is used to set a register in the network being "called". The LIFT action allows a register in the level immediately above the current network to be set. If the ATN has a global HOLD register, a HOLD action sets this register to a node tagged by its constituent type.

Winston [Winston 79] [Winston and Horn 81] describes an ATN as consisting of a grammar and a dictionary. The dictionary portion of an ATN is the knowledge base used in parsing and generating sequences. The term dictionary is used for knowledge base because ATN's have been used almost exclusively for natural language systems. The dictionary is used to store information about the "words" which are recognized by the ATN. In natural language ATN systems, each dictionary entry will consist of a word and its lexical categories. In the Troup and Walter

[Tropf 83] ATN which was designed for object recognition the dictionary consists of structural knowledge and geometrical models and formulas. In this thesis the terms dictionary and knowledge base will be used interchangeably.

An ATN grammar describes a language by representing in a formal way the acceptable sequences of words in the language defined by the network. If, however, the ATN is to be used for analysis, an ATN interpreter (parser or generator) must be written which will take as input the grammar and an input string and produce some resulting structure. Because the grammar is separate from its parser, almost any classical parsing algorithm can be adapted to an ATN as long as some mechanism for handling registers, testing conditions and performing actions is provided. The flexibility of the ATN formalism is demonstrated by the variety of parsing strategies which have been applied to the formalism. Classical top down, left-to-right parsers using simple backtracking have been implemented as well as parsers which work in parallel. There have even been ATN systems designed which parse middle-out beginning from *islands* in the middle of the input string and expanding these out to the end of the string. The following will discuss several different interpretation algorithms which have been used in ATN's.

The original ATN parser was written by Woods in 1970. This parser was designed to be more versatile than the simple top-down, depth first strategy described above. It is based on a list of alternatives, each containing all the information necessary to restart the parser at the point at which the alternative was created. This allows the alternatives to be tried in any order - not simply depth first. An alternative is the current input string, the current state, the remaining arcs to be tried from that state, the register list, and the stack which is used to store information from other levels of the network. The alternatives are used to remember the remaining arcs to be tried, to deal with an ambiguous input word, or to handle ambiguities in the next input word. This parsing scheme was used in the LUNAR system. The LUNAR parser consists of four basic

functions: `PARSER`, `LEXIC`, `STEP`, and `DETOUR`. `PARSER` is called with an input string and it sets up the initial configuration - the initial state, the empty register list and the stack. It then calls `LEXIC` to perform lexical analysis of the input string in order to determine the next word. `PARSER` then calls `STEP` to set up new configurations from currently active configurations. `STEP` uses its arguments (either a configuration or alternative) to follow an arc thus continuing the parse. If there are no more arcs to follow, `NIL` is returned. If `PARSER` runs out of active configurations, `DETOUR` is called to select an alternative to try. Since actions on the arcs can influence the order in which alternatives are chosen, it is possible to reduce the search space necessary to find the most likely parse of the input.

Most ATN parsers have been designed to do left-to-right parsing of an input sentence. The ATN formalism has been demonstrated to be more versatile than that. It has been used by Bates in a very different way in a speech understanding system. This parser operates under the following hypothesis:

...it is not necessarily the case that a speech understanding system should attempt to process an utterance left-to-right, it may be better to begin in the middle with a reliably identified long content word and work from the inside of the sentence out to the ends. In this way, the grammar can also provide predictions about what can be adjacent to a portion of the sentence already processed, and these expectations can be used by the rest of the system in its analysis of subsequent (or previous) portions of the utterance. [Bates 81]

The parser can begin at any point in the input string and parse despite the lack of certainty as to the exact nature of the words at any place in the input. A table holds all partial parse paths for use in another, more complete parse path. The parser also makes predictions about the lexical class of the gaps between sequences. Because the parser works out from *islands* in the data, the arcs in the grammar are indexed to allow easy retrieval of all arcs consuming a particular type of word. Using this indexing scheme, when a word of a particular class is found, all arcs corresponding to that class are retrieved and a corresponding partial parse path is set up in the table for each arc. The parser adds onto these *islands* in the table, working out until the entire utterance has been spanned.

The ATN formalism is not only flexible enough to be used with rather unorthodox parsing strategies, it can also be used to generate syntactically correct strings of a language. The formalism lends itself to this generative power because the grammar is written in a form independent of analysis or production. An ATN generator takes as input an ATN grammar and dictionary and produces sentences. In the simplest case, the generator will begin in the initial state and randomly select an arc leaving that state. If the arc is of type CAT, WRD, or MEM, the generator will attempt to select a word of that category from the dictionary satisfying the conditions on the arc. The following of PUSH, POP, VIR and JUMP arcs is the same as in a parser. Operating in this simplistic fashion, the generator will produce sentences which are only grammatically correct.

2.3. SurfsUP

The following is a brief overview of SurfsUP, the Surface System of the University of Pennsylvania, [Radack, et al 84]. SurfsUP is a collection of data structures and subroutines operating on these structures which may be called directly or be accessed through the interactive front end, IntSurf, provided by the system. This modelling system was chosen to implement the surface and scene models of the image because of the rich data structures, extensive set of accessing and manipulation routines, and the relative ease with which the system can be extended.

The basic data type in SurfsUP is the polysurface or psurf. A psurf is a collection or list of connected surfaces, csurfs. A csurf is a connected network of vertices, edges, and faces. In other words, a csurf is a connected object. A connected object is one in which any node (vertex, edge or face) may be reached from any other node in the object by traversing a sequence of edges belonging to that object network. The data structure permits an arbitrary number of edges and vertices on a face, an arbitrary number of faces on an edge and an arbitrary number of edges impinging on a vertex. There are a few constraints on the representation of the various primitives

(psurfs, csurfs, faces, edges, vertices). An edge must be represented by two vertices. Faces must be convex. This does not reduce the types of polyhedral objects which may be represented since adjacent planes may be coplanar. Also there is no convexity restriction on the shape of csurfs or psurfs.

SurfsUP also provides the basic operations which allow the programmer to manipulate the psurfs without concern for the details of the data structures. Among these basic operations are creation, iteration, and attribute calculation.

The system provides the ability to create instances of the various data structures, Psurfs can be read in from textual files which contain vertex locations and information about the edges and faces which join the vertices. The psurfs may be copied, transformed geometrically, and transformed topologically. Faces may be created using the routine MakeFace which requires a list of edges suitable for defining the boundary of a face.

The iteration routines are extensive, allowing for iteration through the edges or vertices of a face, the vertices of and faces on an edge, and the edges containing a vertex. Also provided are the capabilities of iterating through the vertices adjacent to a particular vertex, the edges connected to a particular edge, and the faces adjacent to a given face.

At a high level, operations are provided for the revolution and spherical sweeping of psurfs, pairwise face intersection between all faces in a psurf and between faces in two or more psurfs, finding the exterior boundary of a psurf, applying a transformation to all vertices of a psurf, testing for inclusion of a point in a psurf, and combining psurfs into hierarchies for representation of complex objects. The system has routines for calculating various metric properties of a psurf such as the volume, area, center of gravity, principle axes, and moments. It also allows the programmer to call built in routines to calculate the direction of the edges in a face and the face

normal. Routines have been provided to split faces and edges - routines which will be necessary since the input data to psurf from the proposed system will be image data. Therefore it cannot be guaranteed that this data will define convex faces, a necessary requirement for SurfsUP. Having these routines will facilitate the manipulation of the data for representation by SurfsUP.

For further explanation of SurfsUP see the *Nasa Project Programmers Guide* [Radack, et al 84] and *A Geometric Investigation of Reach* [Korein 84].

CHAPTER III

Object Recognition

In order to do any high level image processing - ie. recognizing objects and assigning labels to them, it is necessary to have two types of information available to the scene analyzer: the actual visual data and world knowledge enabling the system (be it human or machine) to recognize the possible objects in the domain. These two types of knowledge can take a variety of forms. The actual data can be pixels, straight line segments, circular arcs, corners, regions, generalized cylinders or any other of the many primitives used in image processing to approximate solids. The world knowledge can be represented by production systems [Rosenthal 81]; rule-oriented object hypothesis [Reynolds, et al 84]; CAD type descriptions [Tropf 83]; a combination of schemata, semantic nets, mapdrawings and human interface [Glicksman 83]; or frames [Hwang 83]. I have chosen the ATN formalism to represent these two types of information and to drive the recognition process in LandScan. The ATN design presented here is composed of three parts: the grammar, a dictionary, and the interpreter. The grammar represents the *a priori* or world knowledge that the system must have in order to assign "cultural" labels to subsets of the scene. This world knowledge is what enables the recognizer to parse the scene (which at this point is represented as a set of faces) into subsets which represent known objects. The dictionary is simply a list of all of the faces which comprise the scene and their attributes. It represents the actual data which will be used in the recognition process. The third component of the recognizer is the ATN interpreter. This particular interpreter is a generator rather than a parser. It begins in a designated starting state and traverses the arcs attempting to match the arc with data from the

dictionary. If it successfully reaches a final state, then an object in the scene has been recognized and is added to the current scene model. The following sections will justify the use of an ATN to perform object recognition and discuss the three components comprising the recognizer - the world knowledge which is represented by an ATN grammar, the visual data or dictionary, and the ATN interpreter which drives the recognition strategy.

3.1. Justification for the Use of an ATN

The goal of the LandScan system is to perform query driven analysis of urban scenes. This places two constraints on the object recognition process: it must have a top-down control structure, finding only those objects references in the query, and must encode global knowledge about a domain in which objects of the same type may have very diverse appearances.

The entities found in an urban scene fall into several general categories - buildings, streets, sidewalks, vehicles, and fields to mention a few. Although the objects in the domain are known, their appearances cannot be precisely predicted. Therefore the use of a prototype object to represent global knowledge of the object will not facilitate recognition. It is very likely that most of the actual objects will deviate from their prototype model. Hwang et al. [Hwang 83] and Glicksman [Glicksman 83] both suggest representations of global knowledge for domains of this sort. Hwang et al. argue that a frame based representation system is good for a domain in which the appearances of objects are diverse because "slots" in frames can have default values and thus not have to appear in the actual object. However, the frames still indicate an "ideal" or "prototype" object. The use of a prototype representation is better used in a system where the objects to be recognized are more precisely known - ie. looking for specific objects in a domain such as a machine shop. Glicksman [Glicksman 83] has the world knowledge represented as schemata which encode the knowledge of objects and map drawings of the region to be analyzed. The map definitely captures world knowledge, yet, we are able to analyze scenes without the aid

of a map. Therefore it would be nice to recognize objects in a scene without the use of an auxiliary map. Thus neither of these representation schemes are appropriate for the LandScan system.

Another alternative is the production system which has been used successfully in several different image understanding systems where objects of the same type may have rather diverse appearances [Rosenthal 81]. A production system attempts to match the conditions of a rule to the actual data. For every set of conditions which match, the associated actions of the rule are invoked, thus changing the state of the database. This bottom-up process continues until all possible objects have been recognized or a dead end is reached. The rules in a production system are similar to a grammar in that they can be used to define a sequence of actions to take to recognize a particular object.

The production system was not chosen for the object recognizer because it does not have a top-down flow of control. When the production system interpreter is given a set of rules and the initial configuration of the data base (strictly visual data) it reconfigures the data base by matching the conditions of the rules and then performing the actions of the rules until a final database configuration has been reached. Thus all recognition is performed at one time. While a one time recognition phase would indeed allow the system to respond to questions and reason about the scene, these queries would not be driving the analysis.

The rules of the production system are a good representation of the *a priori* knowledge of this domain, it is the control strategy which does not accurately represent the recognition process performed by LandScan. The ATN formalism enables the global knowledge to be encoded as a generative model for constructing objects from the primitives in the scene while driving the recognition in a top-down fashion. This means that an object is not searched for until its existence is directly addressed in a query.

Although a grammar-like structure adequately represents the global knowledge in this domain, the choice of an ATN grammar to drive the recognition of objects still seems an odd one. After all, the ATN formalism has its roots in formal language theory and was designed for use in natural language systems. The use of an ATN to drive recognition of objects in a visual image is not without precedence. A notable exception to the use of ATN grammars for natural language understanding is the system designed by Tropsch and Walter [Tropsch 83] which uses an ATN model for the recognition of 3D objects with known geometries. The recognition process performed by their system is one of "analysis-by-synthesis" in which hypothesis about an object are generated and then verified by the ATN. The system first generates hypothetical model instantiations, also called prototypes. The prototype or geometric model of an object is a single CAD-like geometric description of that object. These prototypes are then compared to the actual data (the 3D image) using the ATN. The ATN is the generative model. It describes several possible model-to-pattern primitive association sequences. If the similarity between the prototype and the image exceed some threshold then the prototype is considered to be a model instantiation of the actual data. Otherwise, another prototype is generated and matched against the data. The recognition which is controlled by the ATN structure proceeds in the following fashion. First it locates the object in space. The CAT arcs in the system are associated with the placement of the image primitives in 3D space. As more and more primitives are found, the degrees of freedom of the object become more constrained, thus fixing the object in 3D space. Once the object is located, the ATN verifies the object type by testing the pattern primitives of the model against those of the data.

Although Tropsch and Walter use an ATN for object recognition, there appears to be a fundamental problem with the use of this formalism to describe pattern sequences for object recognition. This is the inherently linear ordering placed on the scanning of input. While this linear handling of linguistic data makes sense - we read or hear words in a linear sequence - its use in handling visual data does not necessarily seem justified. We do not obtain the data from a

scene one "element" at a time nor is it likely that we match the features which we have learned to associate with an object in a specific order. Instead, it is likely that we match on "prominent" features in the visual data.

This criticism cannot be dispensed with easily. In order to use the ATN formalism, the recognition of objects must proceed in a sequential fashion, matching the data in the scene to the arcs which define the grammar in a specified order. This sort of recognition scheme seems counter-intuitive to the way in which humans identify objects. The problem is that we most likely match on some feature from the data which appears to the visual system as "prominent". This detection of "prominence" may be biologically hard-wired into the system (ie. a special cell which fires only under certain stimulus) or be learned. Nevertheless, it is difficult, if not impossible, to model this into the system currently being designed. Since we cannot define what a prominent feature in the data is (say nothing of extracting it) we have to fall back on defining an object in terms of the pattern of image primitives which defines it. Allowing the global knowledge to specify the order in which object primitives are matched is not unreasonable since any implementation of the recognizer will place some ordering on the matching sequence. In particular, the ATN grammar presents a straightforward description of the steps which must be followed in order to construct a specified object type from the data primitives present in the actual scene. The ATN formalism has another advantage: the grammar and interpreter are separate. Therefore at some future date a different interpreter which perhaps would match on "prominent" features first (much like the Bates speech parser) could be implemented.

Now that the use of an ATN has been justified, it remains to demonstrate that an augmented transition network is necessary and sufficient to perform object recognition. Both the necessity and sufficiency of the ATN formalism are easily shown. Sufficiency is inherent because an ATN has all the power of a Turing machine. It can represent any language whose sentences

can be generated by a deterministic computational machine. By sequentially matching model features against actual features in the data (in a giant nested IF statement) an object will be recognized. Therefore the ATN is sufficient. The question still remains - might it not have too much power. Johansen et al. [Johansen, et al 83] have also shown that a deterministic finite automata (dfa) which is equivalent to a regular language can be used to detect structure in polyhedra. However, their dfa is a parser which takes as input a string of geometric primitives encoded in a specific order and simply determines whether or not this sequence actually represents some polyhedral structure. It does not actually build an object instance nor are the polyhedra defined in terms of simpler polyhedra. Therefore, a dfa will not capture the two level description of the objects in the urban domain nor is it capable of building object instances. But perhaps a recursive transition network is all that is necessary. The RTN is not powerful enough to do what the system proposes to do - recognize and represent 3-D objects and the relations between them. An RTN does not have the capability of building structures, a necessary feature in the system. The lack of structure building facilities in an RTN is not its only inadequacy in this application. The tests on arcs are also used in recognizing objects. Consider the following example. The ATN has been told to find a building. It first locates a simple building. It then finds another simple building in the data base which is attached to the first building in a manner which suggests that the building is probably a series of rowhouses. The new building is added to the OBJECT register and the SUBTYPE register is set to ROWHOUSE. However, if the SUBTYPE register were already set to COMPLEX, this would indicate that the object being recognized could not be a ROWHOUSE but would fall into the more general category of COMPLEX buildings. If this feature testing were not available, as soon as a simple building were found which was connected to the object in the OBJECT register in a way which suggested rowhouses, the ATN would assume erroneously that it had now found a ROWHOUSE.

Thus the ATN formalism has been shown to be a suitable paradigm for object recognition in

this domain because the grammar describes in a reasonable way the world knowledge necessary for recognizing similar objects having very diverse appearances. It has also been shown that if the world knowledge is encoded as a grammar then a formalism with the power of a Turing machine is necessary for the recognition processes required by LandScan.

3.2. The Object Recognition Grammar

The ATN grammar for object recognition is a set of states and the arcs between them. This grammar represents the world knowledge necessary to enable object recognition. The grammar is a generative model describing in a sequential manner the set of faces and the relations between these faces which must appear in the surface model in order to recognize a particular object. For example, two planes which are contiguous, a certain height from the ground, and having approximately opposite surface normals define a simple building. (Remember, currently this system only handles an aerial view of a scene.) The grammar defining this sequence appears in 3-1.

```
(House
  (cat buildface
    (and (surface_normal <> 0) (#_sides >= 4))
    (setr obj *)
    (setrq type house)
    (to House/half)))

(House/half
  (cat buildface
    (and (surface_normal <>0) (#_sides >= 4)
      (CONTIGUOUS (car obj) *))
    (addr obj *)
    (to House/house)))

(House/house
  (pop))
```

Figure 3-1: A Sample Grammar

The grammar as written is a two level network (this is considerably simpler than most ATN's which handle natural language utterances.) The bottom level concerns itself with the recognition of "simple objects." An object is simple if its further decomposition into parts will result in no entity which is in the domain of objects. For example, decomposing a building with a pitched roof (as described above) will result in two halves of a pitched roof. Neither of these entities are considered objects in the domain - they are parts of objects. This level consists of the networks SIMPBUILD, SIMPSTREET, SIMPFIELD, and SIMPSIDEWALK. The top level combines surfaces and simple objects which were recognized in the first level of the network into "complex objects". A complex object is decomposable in a nontrivial way into at least one simple object. For a complete listing of the grammar see Appendix A.

In the current system this grammar is written following the usual convention where a state is represented as a list: the first element of the list is the state name and the remaining components represent the arcs leading out of that state. The states are named following the Bates' convention of two part names. The first part of the name indicates the name of the network and the second part describes either how far along this state is in the recognition process or the subtype of the object being recognized. The network name is one of the general categories of objects to be recognized, a top level network, or a simpler instance of one of these categories, a bottom level network. Currently the system is recognizing four major types of objects: buildings, streets, sidewalks, and fields. Each of these major categories is divided into subcategories - some of which correspond to complex objects, others to simple objects. See 3-2 for a listing of the major object types and subtypes.

The arcs are represented by lists in the manner described in Bates. The first element of the list indicates the category or type of arc and the second element in the list is dependent upon the type of arc it is. It could be a syntactic category - words or lists of words, a constituent type, the

BUILDING	STREET	SIDEWALK	FIELD
simple	simple	simple	simple
house	laned	curved	parking lot
pyramid		complete	
tiered			
courtyard			
complex			
rowhouse			

Figure 3-2: Major Types and Subtypes of Objects

next state, or the form in which the data "parsed" is to be returned. Next is a list (possibly empty) of tests to be performed before the arc can be traversed. Unless the arc is a POP arc there will follow a list of actions to be performed as the arc is traversed. Finally, the next state is specified (for all arcs except JUMP and POP types).

The labels on the arcs (object type or surface type) represent the type of components which will form the object either a face or a simple object. The tests on the arcs encode the relations which must hold between the components of an object and also provide further checking of the features of a component. Although a building may be combined with a surface thus producing a complex building, this may not be possible if the building is not of the correct subtype. A condition testing the subtype register handles all such cases. The relations between faces are tested by inspecting the corresponding entry in the face relation tables.

The actions performed on the transitions or arcs of the ATN are the standard register setting actions of all ATN's. There are two registers associated with the system - a SUBTYPE register and an OBJECT register. The SUBTYPE register contains the current subtype of the object being recognized. It is a feature register whose value is chosen from the set of subtypes found in 3-2. This subtype can change as different parts of the object are matched. The OBJECT register is a role register containing a list of all the faces which comprise the object. As faces are found

which match the generative sequence described by the grammar, they are added to the OBJECT register. Once an object has been fully recognized (a final state in a top-level network has been reached) it is added to the knowledge of the scene. The addition of another object to the scene model is performed by the BUILDQ action. This action only appears on the arcs leaving states in which a complete object has been identified.

3.3. The Dictionary

The dictionary in the recognizer is the surface model which represents the visual data, all of the faces in the scene and their attributes. The surface model represents both the geometric and topological information about the surface primitives in the scene.

The geometric attributes associated with each surface or face are the area of the face, its surface normal, the centroid, the shape, and the compactness of the face. The area and shape of the face are easily computed. The shape is determined by the number of edges bounding the face. The centroid is the height from the ground of the center of the face. The centroid was chosen to represent height because many faces will not be horizontal. If the face is not horizontal, the height of the face (or distance from the ground) is not immediately apparent. The centroid appears to be a good approximation of this. The surface normal represents the angle of the face. Many faces will not be horizontal and the angle of the surface is instrumental in the recognition process. The compactness is a further representation of the shape of a face. Just knowing the number of sides of a face is not always an adequate representation of its shape. Many objects in the urban scene may be approximated by a rectangular face - a face bounded by four edges. For instance, both a sidewalk and a street are represented by quadrilaterals and both can have the same centroid and normal. Yet, they should not be confused - the surface representing the sidewalk is not as compact as the face for a street - the sidewalk is narrower.

The relations between the faces represent the topological properties of the faces. In order to do recognition four topological relations are necessary. They are ADJACENT, CONTIGUOUS, ABOVE, and CONTAINS. While ADJACENT and CONTIGUOUS seem to represent the same relationship, the CONTIGUOUS relation is a much stronger condition. In fact, CONTIGUOUS is a subset of ADJACENT. Two faces are considered ADJACENT if they share at least one point. Two faces are CONTIGUOUS if they share at least two points - in other words, they share a segment. The ABOVE relation does not represent the normal intuitive meaning of ABOVE. A face is considered ABOVE another face if the centroid of the first face is strictly greater than the centroid of the second face. The intuitive meaning of above usually includes some notion of proximity as well as difference in height. This, however, would have been too difficult to capture in the representation. The CONTAINS relation means that one face is completely surrounded by another face when they are both projected onto the xy-plane.

These topological relations are represented by adjacency matrices - one matrix per relation. The "nodes" in the graph represented by the adjacency matrix are the faces which have been found in the scene. The matrix is an $n \times n$ boolean array where 0 corresponds to no relation between faces and a 1 to the relation holding between them. None of the relations are reflexive. ABOVE and CONTAINS are transitive, ADJACENT and CONTIGUOUS are symmetric.

For a more complete description of the surface model the reader can see *Construction of a Three Dimensional Surface Model* [Krotkov 84].

3.4. The ATN Interpreter

Unlike most ATN's (in natural language understanding as well as other applications) which have been designed to parse an input string, this system will not have an object which it wishes to parse into its components in order to determine whether or not it is a valid object. This ATN interpreter operates as a generator taking a grammar and a dictionary as input and producing strings as output. In the case of natural language generation, the output string is a sentence. In object recognition, the output is an object instance. The "dictionary" which will be used is a list of all the surfaces which have been found in a bottom-up fashion by the low and middle level image processing routines. The generation control structure works as follows. The control begins in some initial state (determined by the object which is being looked for). The generator then selects (somewhat randomly) an arc to follow. If a face (or other specified structure) is found to be a subset of the list of faces (dictionary) then the chosen path in the ATN is continued. The generation is complete when a final state in a top-level network (one with a POP arc) is reached. At this point a new object is added to the object data base, and the surfaces used in the building of the object are marked as used to avoid using the same face in more than one object.

The registers are stored as an association list: a list of pairs. The first element of the pair is the register name and the second element is a pointer to its value. The register structure has been designed this way for two reasons. First it allows the ATN to function recursively without losing the registers which had been set earlier. Secondly, the <name pointer_to_value> structure allows the actual registers to be implemented in SurfsUP. This is a necessity since all the registers will hold SurfsUP data types and these data types are not easily represented in LISP.

The states are stored as a symbol with a property list. The state name indicates the name of a symbol having the property ARCS. ARCS is a list of all the arcs for which that state is the initial state. This storage of the state allows the grammar to be separate from the interpreter. It

is read in using a `DEFINE_STATE` function which sets up the property list for each state. The property list representation also facilitates the retrieval of the arcs associated with a state. The retrieval process is done with a simple `(GET STATE 'ARCS)` function call.

The arc structure is of the form `(TYPE HEAD TEST ACTION*)`. `TYPE` specifies the kind of arc - `PUSH`, `POP`, `CAT` or `JUMP`. These are the only arc types necessary for this grammar. The `CAT` action is performed by a `SurfsUP` routine which searches the surface model for the type of arc indicated by the arc label. `PUSH`, `POP` and `JUMP` alter the flow of the LISP code without calling any external routines. `HEAD` specifies additional information which must be known in order for the arc to be taken. If the `TYPE` is `CAT` the `HEAD` will be the category of the surface, `PUSH` and `HEAD` is the `ATN` which is called, `JUMP` - the next state in the `ATN`, `POP` and `HEAD` is the `<FORM>` of the constituent to be returned. There are three types of `FORMs` in this system:

- * - refers to the current item of input. In the actions on a `PUSH` arc it is the value of the constituent returned.
- `(GETR REG)` - returns the current contents of register `REG`. In this system only this `FORM` will appear as the `HEAD` of a `POP` arc.
- `<OBJECT_TYPE OBJECT SUBTYPE>` - explained below.

`TEST` is the conditions which must further be met in order for that arc to be traversed. The tests allow the grammar to be context sensitive. Often tests involve checking the contents of registers which have been previously set. `ACTION` specifies the register setting and structure building to be done as the arc is traversed. The possible actions are:

- `(SETR REG VALUE)` - sets the register `REG` to the evaluation of `VALUE`
- `(SETRQ REG STRING)` - sets the register `REG` to the literal `STRING`
- `(ADDR REG VALUE)` - appends the evaluation of `VALUE` to the end of the list in `REG`
- `(BUILDQ <FORM>)` - builds a constituent of the structure specified by `<FORM>`. The constituent built in this system will be an object instance. In this system there is only one `FORM` used in a `BUILDQ`: `<OBJECT_TYPE OBJECT SUBTYPE>`

- OBJECT_TYPE is a major object type as in 3-2
- OBJECT is the OBJECT register
- SUBTYPE is the SUBTYPE register

These actions are actually performed by the SurfsUP routines since the surface model has been implemented in SurfsUP. The SETR and SETRQ are simple Pascal assignment operations. ADDR is implemented by the SurfsUP routine AddFaceToList in which the face is added to the OBJECT register which holds the set of faces composing the object. BUILDQ calls a series of SurfsUP routines which add the newly found object to the scene model.

The control structure for the ATN is provided by a generator which is a series of LISP functions. The first function GENERATE is called with one argument - the starting state of the grammar. The initial configuration (initial state, register list, stack) is set up by this call. A function ATN is then called with the starting state. ATN is the function which selects the arc which is to be followed. The backtracking is a simple, depth first strategy. If the first arc fails then the next arc in the arc list associated with the state is called. From ATN the EVALARC function is called with the arc to be evaluated and the association list representing the set of registers. First EVALARC determines the type of arc which is being considered as a possible transition. Once the type of the arc has been determined, the function associated with that arc is performed. If that function returns a non-nil value - in the case of a CAT or PUSH arc - TEST is evaluated to see if the next state can actually be reached. The tests must be performed after the CAT or PUSH actions because the ATN is performing generation. This means that the actual data is not known until after the HEAD is matched. This differs from parsing where the current input is available before the arc is traversed. In generation the CAT and PUSH arcs will return new structures which are then tested against other already existing structures and registers. If the arc is a JUMP or POP then the tests must be performed before the JUMP or POP action is

attempted. Finally, if all the tests are true, the actions are evaluated by the EVALACTIONS function and the ATN enters a new state. This new state is determined in one of three ways. If the arc is a JUMP arc the HEAD of the arc list structure specifies the next state and ATN is called with HEAD. A POP arc will call the function POPATN which either returns the flow of control to the arc from which the network was invoked or determines that a final state in the top-level network has been reached and thus add a new object instance to the model. Any other type of arc will have a <TO STATE> action as the last action in ACTION* which calls the TO function in the generator. The TO function then calls ATN with this new state and the traversal continues in this fashion until a final state is reached or the functions gets halted in a non-final state with no more arcs to try.

CHAPTER IV

Representation - The Scene Model

Simply recognizing objects is not adequate for a system which is designed to analyze a scene. The scene must be represented in a manner which facilitates the analysis. At all stages of image processing the scene is represented - as pixels, edgels, edges, faces, or objects (these are the various levels of image processing in this system). However, not all of these representations are conducive to high level scene understanding. Imagine having to answer a question about the existence of an object when the only representation of the visual data was at the pixel level. In order to address the query, the system would have to have some way of grouping together a certain number of pixels of approximately the same grey level, calculating the shape of this region, etc. All of these steps would have to occur each time a question was asked. Not only would such a representation add enormous amounts of overhead to the scene analysis, the representation reflects none of the intuitive knowledge of the objects in the scene and the relationships between those objects. Therefore, in order to perform any scene analysis in a reasonable way the scene has to be represented in some fashion which will enable the operations necessary for scene analysis to be performed. These actions include among others calculating of relations, both complex and simple, among objects; locating specific features of objects; and identifying specific types of objects. The representation of the scene - the scene model - described below has been designed to facilitate these operations necessary for robust image understanding. Since one of the goals of the system is to understand locative constructs in natural language utterances, special detail has been paid to representing the primitive spatial relations between

objects in a way which will enable the system to understand phrases expressing more complex relations between objects. The following sections will discuss the overall strategy used in modelling a scene, the representation of object instances, the method of encoding the relations between objects, and the implementation of the model constructor.

4.1. The Basic Design Criterion

Various representation schemes have been employed in image understanding systems and no optimal object representation has yet been found. The visual primitives used to represent objects are highly domain dependent. In a 2D domain polylines or concatenated line segments, fourier descriptions, conic sections, B-splines, strip trees, or regions have been used to describe objects. In 3D the objects can be represented by a surface or boundary, generalized cylinders, or some volume measure (constructive solid geometry). The final choice of representation will depend on the dimension of the domain, the types of objects in the domain, the scene analysis operations which will be performed on the final representation, and the manner in which the representation is to be obtained. The 3D MOSAIC system [Herman 83] has a dynamic scene description which accumulates scene information as different views of the image are processed and can be used to perform various scene analysis tasks. The objects are represented graphically with the nodes being object primitives - faces, edges, and vertices - and the arcs being the topological relations between these primitives. The representation of objects by their surface primitives was chosen because it is compact and easy to analyze. In other systems the scene is represented by frame instantiation [Hwang 83], or schemata instantiation [Glicksman 83]. These two encodings can be used when the features of the scene are more defined in advance. Rosenthal represents objects in a scene using what he calls an Object Description Notation [Rosenthal 81]. In this notation objects are represented by LISP lists made up of three components: visual information (color, shape, etc), contextual information (the likely spatial relations between objects), and the

regularity in spatial relations (the repetitive spatial relations an object has with itself and other objects).

Most of the queries made about the scene will address the existence of objects and object parts, and the spatial relations between objects. Therefore the representation must facilitate the finding of objects in the scene as well as the analysis of the spatial relations between objects. This dictates that the object representation encode in it both objects in the scene and the primitive relations between objects. These primitive relations must be selected and represented in a way which enables LandScan to easily compose them obtaining the more complex relations used in the scene analysis. The scene model also has to be easily constructed from the data which is used in the recognition process. Finally, since the analysis is query driven, it is only necessary to have those parts of the scene currently of interest to the user will be represented in the symbolic scene model. As the areas of interest in the scene may change over time, the scene model must reflect this flux. Therefore a dynamic scene model was designed which is composed of two components: a list of objects currently known to be in the scene and a set of matrices representing the primitive relations which have been found necessary and sufficient for performing further scene analysis.

The scene model is dynamic because information can be added to it as further image analysis occurs. When the scene analysis begins - (ie) when the system is presented with an image it has never processed before - the scene model is empty. The other models of the image which have been constructed by data driven processes are automatically generated as soon as the image is processed. If no queries are made of the system, then no scene model is ever created. When a question is asked, the scene analyzer/constructor searches for the entities whose existence is in question using the object recognizer described above. As soon as the objects queried are found they are added to the list of objects known to be in the scene. The primitive relations between

entities must then be updated - the relations are now defined over a superset of the old set and all appropriate new tuples must be added to the newly extended relations.

Keeping a list of objects known to be in the scene allows the addition of further information to the scene model to be a trivial task. The object list component of the scene model is the set over which the primitive spatial (topological) relations is defined. Therefore adding the new tuples to the relations will only involve calculating the relations between the new entities and the new set over which the relations are defined. Thus the choice of dynamic model is feasible and will allow for a top-down scene analysis.

4.2. The Object List

The first component of the scene model is the object list. The objects on this list are those objects which have been recognized during previous scene analysis operations. These objects are represented by polyhedral surfaces. Thus to the high level reasoner it appears that objects are composed of only bounded planes - primitives at one level of representation - the edge and vertex representations are not accessible to the reasoning processes. This differs from the 3D MOSAIC system [Herman 83] where objects are represented by faces, edges, and vertices. Conceptually, these primitives are at different levels of the processing because edges are sets of vertices and faces are sets of edges. The use of a single primitive (or a set of primitives which are not composed from one another) is conceptually cleaner to work with and is adequate for modelling objects. Each instance of an object in the scene has the information associated with it which was determined necessary to facilitate further scene analysis. The components of an object record are a name, the list of faces comprising the object, its location in Euclidean three space, and an indication of the subtype of the object. The name of the object indicates the type of the object. This name is one of the objects which can be expected in the current domain. The indication of the subtype gives more specific information about the object - the expectations one can have

about an object when analyzing a scene. For example, the name of an object might be building while the subtype is house. The face list represents the set of polyhedra which comprise the object. The faces on the list are not stored with any indication as to what part of the object they correspond. This is unnecessary since the objects will not be composed of large numbers of faces. Therefore, any part analysis operation can search through the list until it finds the face which is likely to correspond to the part. The final bit of information associated with an object is its location in Euclidean three space. For lack of any better approximation of location the centroid of the object is used to represent the object's metric location. The problem with assigning a metric to location is that an object occupies many points on the grid and it is difficult to say which point best describes the location of an object. The centroid of an object in this system is defined to be the average of all the centroids of the faces on the face list of that object.

4.3. The Relations

The relations in the scene model represent the primitive relations or topological properties between objects in the scene. The same four relations as in the surface model are adequate to represent all relations, both simple and complex, among objects using various forms of relational composition. These four relations are ADJACENT, CONTIGUOUS, ABOVE, and CONTAINS and are defined over the set of all objects currently recognized in the scene. The relations are represented by their adjacency matrices because the adjacency matrix is easily updated and makes composition of relations simple. The composition becomes a simple matter of boolean matrix multiplication for which there are many fast and efficient algorithms.

The definitions of the four relations are very similar to those of the surface model. CONTIGUOUS is a subset of ADJACENT. Two objects are said to be ADJACENT if they satisfy the following condition:

$\exists \text{FACE}_1 \in \text{OBJECT}_1$ and $\exists \text{FACE}_2 \in \text{OBJECT}_2$
such that FACE_1 ADJACENT FACE_2

OBJECT_1 is CONTIGUOUS to OBJECT_2 if:
 $\exists \text{FACE}_1 \in \text{OBJECT}_1$ and $\exists \text{FACE}_2 \in \text{OBJECT}_2$
such that FACE_1 CONTIGUOUS FACE_2

Once again, CONTIGUOUS is a subset of ADJACENT. These two relations are only symmetric. The calculations of CONTIGUOUS and ADJACENT are expensive since finding each one of these relations for two objects will take $O(mn)$ where OBJECT_1 has m faces and OBJECT_2 has n faces. However, the calculations will consist in checking the CONTAINS and ADJACENT relations of the surface model rather than recalculating the boundaries of the actual surfaces. This will reduce the actual operations involved in determining the ADJACENT and CONTIGUOUS relations.

The ABOVE relation is computed by performing a simple comparison of the location fields, centroids, of the two objects. If the centroid of OBJECT_1 is higher from the ground than the centroid of OBJECT_2 then:

OBJECT_1 ABOVE OBJECT_2

Like ABOVE in the surface model, the definition of ABOVE in the scene model is not the intuitive definition of ABOVE which carries with it some notion of proximity. This proximity constraint is not part of the definition of the primitive relation ABOVE.

The CONTAINS relation is the most difficult relation to compute. First the boundary of the face list component of each object must be calculated. These boundaries are then projected onto the xy-plane. The relation is then defined as follows:

$\text{Boundary}(\text{OBJECT}_1) \cap \text{Boundary}(\text{OBJECT}_2) = \text{Boundary}(\text{OBJECT}_1)$
then OBJECT_2 CONTAINS OBJECT_1
 $\text{Boundary}(\text{OBJECT}_1) \cap \text{Boundary}(\text{OBJECT}_2) = \text{Boundary}(\text{OBJECT}_2)$
then OBJECT_1 CONTAINS OBJECT_2

ABOVE and CONTAINS are only transitive.

4.4. Justification of the Scene Model

It remains to show that the proposed scene model both adequately represents the objects in an image and facilitates the successful execution of analysis operations. As mentioned above, most of the analysis operations will fall into one of three categories: determining the existence of an object, finding the part of an object, or computing the relation which represents a locative construct relating objects. The existence of objects will be resolved in one of two ways - either by finding the object in the scene model by searching the object list, or by using the recognizer to find a new instance of the object. To find a part of an object its face list will be searched until the part is found using the global knowledge about parts embodied in the object model. As for resolving the interpretation of locative constructs, the relations allow objects to be located relative to other objects in the scene. Suppose the question were asked, "Is there a car on the street?" An object of type CAR is ON an object of type STREET if the following primitive relations hold:

CONTAINS(STREET,CAR)
ABOVE(CAR,STREET)

The reasoner would determine if the CAR is ON the STREET by calculating the following relation composition:

CONTAINS * ABOVE^T

which would be calculated by a simple matrix multiplication of the CONTAINS adjacency matrix and the transpose of the ABOVE adjacency matrix. So the understanding of relational expressions will be accomplished by composing the primitive relations. The necessary compositions of primitive relations will be determined by linguistic knowledge used by the natural language interface in understanding the queries. Since the model facilitates these three operations which are essential to any scene analysis it, in fact, is robust enough to be used by the LandScan image understanding system.

4.5. Implementation of the Scene Model

The scene model has been implemented in Pascal as part of the SurfsUP system [Radack, et al 84]. As mentioned above, SurfsUP was chosen because of the relative ease in which complex objects which are approximated by polygonal surfaces can be represented. There are two data types in SurfsUP which were considered for representing objects, OBJECT and csurf. However, neither captures the essential structure of an object - a set of faces and attributes. The SurfsUP object is conceptually very different from the LandScan object. Objects in SurfsUP are represented hierarchically. Each part of the object (a psurf) is defined in its own coordinate system. These parts are then related by transformations between the parts. This is a much more complex structure than necessary to model LandScan objects and, more importantly, it would not clearly indicate that an object is a set of faces. A LandScan object is much more like a csurf. The csurf was not chosen for object representation because the csurf is represented conceptually by sets of vertices with the edges and faces contingent upon them as secondary fields in the csurf record. However, this means that the representation of objects in the scene model would not correspond to the representation of objects in the ATN grammar - sets of faces. Since neither OBJECT's nor csurf's really correspond to objects, a new data type was added to SurfsUP - the ENTITY which represents a LandScan object. The ENTITY data structure is a set of faces and attributes. Below is a description of the ENTITY data structure and the relation data structure used to define a scene.

The ENTITY data type was designed to be analogous to the face data type which is the primary structure used in the surface model. This was done to keep the modelling structure clean between all levels. Thus the Scene Model is analogous to the Surface Model except that the components of the image at this level are objects (sets of faces) instead of single faces. Using this representation it is very apparent that the scene model is constructed only from components in

the surface model. An ENTITY record has the following fields: a name which is the general label associated with the object (building, street, etc); a face list; an attribute list - the location of the object and its subtype; and a unique identifier field, id. The id field holds a unique integer between 1 and the maximum number of entities which may appear in a scene. This field must be present to allow distinct entities to be easily distinguished. The set of entities in the scene is represented by a linked list of ENTITYNODES with pointers to both the first and last entities in the list. This makes it easy to add entities to the list (they are pushed onto the end) and also to keep track of the beginning of the list (necessary for performing iteration). An ENTITYNODE consists of a pointer to an ENTITY and a pointer to the next element in the list. For a complete description of the data structures representing entities and the entity list see Appendix B.

Once an object has been recognized, the ATN generator calls a routine to add an entity to the scene model, AddEntityToSceneModel. This routine creates a new entity instance and sets the fields with the contents of the registers OBJECT and SUBTYPE. The new entity record is then added to the entity list - a new entity node is pushed onto the end of the current entity list. When the new entity is created an internal bookkeeping routine assigns the entity its unique identifier, id. After the entity list has been updated, the relational portion of the scene model is updated.

The entity relations, like the face relations, are represented by boolean matrices indexed by the entities in the scene so far. The SurfsUP system knows how many elements are in the scene because it keeps a "marks array" of all possible entities in the scene. If an entry in this marks array is set to true then that object exists in the scene, otherwise it is an entity instance which has not yet been allocated. The entity relation matrices are indexed by the unique identifier id. If there were no scalar way of determining the difference between two entities, it would be conceptually difficult to see the correlation between the entities and the entity relations. A 1 in

the entity relation means that the relation between the two objects holds, a 0 indicates the opposite. Since these relations are dynamic (the entire model of the scene is so) they are declared to be the maximum size allowable, MAX_ENTITIES_WORLD x MAX_ENTITIES_WORLD. This allows new tuples to be easily added to the adjacency matrices. The routine to update the entity relations works in the following way: The new entity is compared to each entity in the entity list. Using the properties of the relations (see above) the relations are updated accordingly.

This is the implemented representation of the scene model and a brief overview of how the actual construction occurs. A complete listing of the code to perform the scene model construction can be found in the TEMPUS library.

CHAPTER V

Future Work

The recognizer and scene model will provide the image information necessary to perform scene analysis of urban environments. In order to provide a robust scene representation for the natural language driver, several extensions must be made to this work. The grammar must be extended to represent other objects which also occur in urban settings, the world and object models must be fully defined and encoded for use by the natural language understander, a natural language interface must be designed, and the recognizer must be fully implemented.

Currently the grammar describes the construction procedure to follow to recognize four classes of objects: buildings, streets, sidewalks, and fields. The scene must be viewed from above in order to recognize these objects. The first extension which must be made to the ATN grammar is the addition of rules to enable the recognition of other objects appearing in urban environments - vehicles, people, trees, etc. With the current grammar the scene understander can only be queried about the large bodies in the picture - buildings, streets, sidewalks and fields. It is unlikely that users will only query about these four kinds of objects. The addition of other object definitions to the grammar will provide a more realistic and robust image analyzer. Ideally, the system should also be able to reason about scenes from perspectives other than the aerial one. I believe that the grammar proposed can be extended to encompass other views by the addition of rules describing the construction of objects from other views.

Before the scene model can be used in the image analysis process, the global knowledge

embodied in the world and object models must be encoded. The world model will be represented graphically as explained above. The object model presents more of a challenge though. The PART_OF relation has been handled in many systems [Rosenthal 81], [Shapiro 84]. However, no one has yet proposed a means of encoding the linguistic data which must be known about the objects in order to use them correctly in natural language utterances. Herskovits [Herskovits 82] suggests that the following object knowledge is relevant to the task of encoding and decoding locative constructions. She proposes that you must know the general type of the object (solid, liquid or gas), its gravitational behavior, and its appearance at various distances. Other attributes which might be necessary are shape, size, characteristic orientation, function, a typical geometric conceptualization, a typical physical context, actions commonly performed with the object, its functionally salient parts, its perceptually salient parts, and the normal relations it might have with other objects. The encoding of this linguistic information will have to be addressed before the scene analyzer can be query driven. With the use of the global knowledge embodied in those two models, the scene model represents the scene data so that it can be analyzed.

The implementation of the recognizer must be completed. Currently, the representation of the scene model has been fully implemented in SurfsUP. The grammar for representing objects has been written but not tested in an actual computer implementation. The ATN interpreter has been defined by not fully implemented. The ATN interpreter must then be linked to the surface model data accessing routines and the scene model constructor. Once this has been done the full recognizer and dynamic scene model will be operative and will be linked to the scene analyzer and natural language interface.

The natural language interface which uses the scene representation still has to be designed. It must be able to apply locative linguistic constructs to some representation of visual data and reason about this data. The natural language interface will be the next major module which must

be designed. When this is operative, the scene analysis will be truly query driven and the goals of the system will have been reached.

CHAPTER VI

Conclusion

A recognition scheme and symbolic image representation, necessary components of a natural language driven scene analyzer, have been proposed.

The ATN formalism has been adopted for the recognition process. This choice was made because the formalism has a top-down flow of control and the grammar describes in a perspicuous way a method for finding the set of primitives which determines an object in the scene. The grammar does so by representing recognition as a sequential process in which faces are searched for in the image. The search is constrained by the features which the faces must have and the various primitive relations which must hold between faces in order for the surfaces to be in the set defining the object. Although a recognition scheme in which primitives must be matched in a specific order seems an odd choice for visual processes, it has been shown that the formalism is applicable to the domain and at some future time, it might be possible to design an interpreter which is not constrained by a left-to-right parsing/generating strategy. The ATN is driven by a generator, thus making recognition a process in which sets of surfaces composing objects are constructed from the surface model. The model has also been shown to be especially suited to a domain in which objects of the same type may have very diverse appearances. An additional advantage of the ATN is that the recognition module will interface easily with high level reasoning processes. The reasoner will determine from the query the objects of interest. It will then call the recognizer and ask it to find those objects. When this has been done, the reasoner will be able to generate the proper response to the question. The recognizer also interfaces easily

with the scene representation constructor. Using the BUILDQ action, objects are added to the scene model. The ATN proposed satisfies the criteria of the LandScan recognizer:

1. It is driven by a focussing mechanism (natural language queries).
2. It interfaces easily with both reasoning and low-level vision processes.
3. It can recognize objects of the same type having diverse appearances.

A symbolic representation for the scene and objects in the scene has been proposed which will facilitate high level reasoning processes driven by goal-oriented analysis. Since the analysis is goal-oriented, we will only want those parts of the scene which are or have been of interest in the scene model. The changing focus means that the scene model will have to be dynamic. The proposed symbolic representation of the scene is constructed as LandScan is queried, thus reflecting the change in focus. Objects are represented by a single image primitive - the polyhedron. This keeps the various representations of the scene hierarchical - each higher level is composed of entities from the level immediately below it. An object in the scene is represented by the set of faces which determine it (this set was constructed by the recognizer), and some features of the object (a metric location, type and subtype). The scene model has two components: a list of objects currently known to be in the scene and a set of primitive locative relations between these objects. The object representation will facilitate operations in which a part of an object is in question. The object list and recognizer will allow the existence of particular objects to be determined. It has been shown that the four primitive locative relations - ABOVE, CONTAINS, ADJACENT and CONTIGUOUS - can be composed to obtain information about more complex relations between objects as embodied in locative constructs. The scene model also fulfils the LandScan criteria:

1. It is dynamic, reflecting the user's interest in the scene.
2. It facilitates reasoning processes:
 - locating an object

- locating an object part
- determining relations, both complex and simple, among objects

Therefore, the recognition paradigm and scene model proposed will facilitate the top-down analysis of aerial images guided by natural language queries.

APPENDIX A

ATN Grammar for Object Recognition

GRAMMAR FOR COMPLEX BUILDINGS

```
(building
  (push simpbuild
    (setr OBJECT *)
    (liftr TYPE *)
    (to building/start)))

(building/start
  (jump compbuild/center
    (or (equal TYPE center) (equal TYPE courtyard)))
  (push simpbuild
    (and (rowjoin OBJECT *) (not (equal TYPE complex)))
    (addr OBJECT *)
    (setrq TYPE rowhouse)
    (to building/row))
  (push simpbuild
    (centerjoin OBJECT *)
    (addr OBJECT *)
    (setrq TYPE center)
    (to building/center))
  (push simpbuild
    (contiguous OBJECT *)
    (addr OBJECT *)
    (setrq TYPE complex)
    (to building/start))
  (jump building/add))
```

```
(building/row
  (push simpbuild
    (rowjoin OBJECT *)
    (addr OBJECT *)
    (to build/row))
  (push simpbuild
    (contiguous OBJECT *)
    (addr OBJECT *)
    (setq TYPE complex)
    (to building/start))
  (jump building/add))

(compbuild/center
  (push simpbuild
    (contiguous OBJECT *)
    (addr OBJECT *)
    (to compbuild/center)))
  (jump building/add))

(building/center
  (push simpbuild
    (or (centerjoin (car OBJECT) *) (centerjoin (cadr OBJECT) *)))
    (addr OBJECT *)
    (to building/cent4))
  (push simpbuild
    (contiguous OBJECT *)
    (addr OBJECT *)
    (setq TYPE complex)
    (to building/start)
    (jump building/add))
```

```
(building/cent4
  (push simpbuild
    (or (and (centerjoin (car OBJECT) *)
              (centerjoin (caddr OBJECT) *))
        (and (centerjoin (cadr OBJECT) *)
              (centerjoin (caddr OBJECT) *))))
    (addr OBJECT *)
    (to building/centtype))
  (push simpbuild
    (contiguous OBJECT *)
    (addr OBJECT *)
    (setq TYPE complex)
    (to building/start))
  (jump building/add
    (setq TYPE complex)))
```

```
(building/centtype
  (cat fieldface
    (contains OBJECT *)
    (addr OBJECT *)
    (setq TYPE courtyard)
    (to building/start))
  (cat buildface
    (contains OBJECT *)
    (addr OBJECT *)
    (setq TYPE tiered)
    (to building/start))
  (push simpbuild
    (contains OBJECT *)
    (addr OBJECT *)
    (setq TYPE tiered)
    (to building/start)))
  (push field
    (contains OBJECT *)
    (addr OBJECT *)
    (setq TYPE courtyard)
    (to building/start))
  (jump building/add
    (setq TYPE complex)))
```

```
(building/add
  (jump building/building
    (buildq ('building OBJECT TYPE))))
```

```
(building/building
  (pop (getr OBJECT)))
```

GRAMMAR FOR SIMPLE BUILDINGS

```
(simpbuild
  (cat buildface
    (equal (sn *) vertical)
    (setr OBJECT *) ;set the OBJECT reg
    (setrq TYPE simple)
    (to simpbuild/add))
  (cat buildface
    (and ((sn *) <> 0) ((nosides *) >= 4))
    (setr OBJECT *)
    (setrq TYPE house)
    (to simpbuild/hali))
  (cat buildface
    (and ((sn *) <> 0) ((nosides *) = 3))
    (setr OBJECT *)
    (setrq TYPE pyramid)
    (to simpbuild/pyrq))
  (cat buildface
    (and ((sn *) <> 0) (equal (shape *) 'ringquad))
    (setr OBJECT *)
    (setrq TYPE tiered)
    (to simpbuild/tiered)))
```

```
(simpbuild/half
  (cat buildface
    (and ((sn *) <> 0) ((nosides *) >= 4)
      (contiguous (car OBJECT) *))
    (addr OBJECT *) ;push face into OBJECT
    (to simpbuild/add)))
```

```
(simpbuild/pyrq
  (cat buildface
    (and (sn <> 0) (equal nosides 3) (contiguous (car OBJECT) *))
    (addr OBJECT *)
    (to simpbuild/pyrh)))
```

```
(simpbuild/pyrh
  (cat buildface
    (and ((sn *) <> 0) (equal (nosides *) 3)
      (or (contiguous (car OBJECT) *)
          (contiguous (cadr OBJECT) *))))
    (addr OBJECT *)
    (to simpbuild/pyr3)))
```

```
(simpbuild/pyr3
  (cat buildface
    (and (sn <> 0) (equal nosides 3)
      (or (and (contiguous (car OBJECT) *)
                (contiguous (caddr OBJECT) *)))
          (and (contiguous (cadr OBJECT) *)
                (contiguous (caddr OBJECT) *))))
    (addr OBJECT *)
    (to simpbuild/add)))
```

```
(simpbuild/tiered
  (cat fieldface
    (contains (car OBJECT) *)
    (addr OBJECT *)
    (setrq TYPE courtyard)
    (to simpbuild/add))
  (cat buildface
    (contains (car OBJECT) *)
    (addr OBJECT *)
    (to simpbuild/add)))
```

```
(simpbuild/add
  (jump simpbuild/simpbuild
    (not (contiguous OBJECT facelist))))
    ;tests OBJECT against all possible
    ;building faces
```

```
(simpbuild/simpbuild
  (pop (getr OBJECT)))
```

GRAMMAR FOR STREETS

```
(street
  (push simpstreet
    (setr OBJECT *)
    (setrq TYPE simpstreet)
    (to street/comp)))
```

```
(street/comp
  (push simpstreet
    (contiguous OBJECT *)
    (addr OBJECT *)
    (setrq TYPE laned)
    (to street/comp))
  (jump street/add))
```

```
(street/add
  (jump street/street
    (buildq ('street OBJECT TYPE))))
```

```
(street/street
  (pop (getr OBJECT)))
```

GRAMMAR FOR SIMPLE STREETS

```
(simpstreet
  (cat streetface
    (setr OBJECT *)
    (to simpstreet/build)))
```

```
(simpstreet/build
  (cat streetface
    (and (contiguous OBJECT *) (*joined along narrow* OBJECT *))
    (addr OBJECT *)
    (to simpstreet/build))
  (pop (getr OBJECT)))
```

GRAMMAR FOR FIELDS

```
(field
  (cat fieldface
    (contiguous OBJECT *)
    (setr OBJECT *)
    (setrq TYPE simple)
    (to field))
  (jump field/field
    (buildq ('field OBJECT TYPE))))
```

```
(field/field
  (pop (getr OBJECT)))
```

GRAMMAR FOR SIDEWALKS

```

(sidewalk
  (push simpsidewalk
    (eq (shape *) 'ringquad)
    (setr OBJECT *)
    (setrq TYPE connected)
    (to sidewalk/comp))
  (push simpsidewalk
    (setr OBJECT *)
    (setrq TYPE simple)
    (to sidewalk/comp)))

(sidewalk/comp
  (push simpsidewalk
    (contiguous OBJECT *)
    (addr OBJECT *)
    (setrq TYPE curving)
    (to sidewalk/comp))
  (jump sidewalk/sidewalk
    (buildq ('sidewalk OBJECT TYPE))))

(sidewalk/sidewalk
  (pop (getr OBJECT)))

```

GRAMMAR FOR SIMPLE SIDEWALKS

```

(simpsidewalk
  (cat sidewalkface
    (setr OBJECT *)
    (setrq TYPE simple)
    (to simpsidewalk/extend)))

(simpsidewalk/extend
  (cat sidewalkface
    ("joined along narrow" OBJECT *)
    (setr OBJECT *)
    (to simpsidewalk/extend)
    (jump simpsidewalk/simpsidewalk))

(simpsidewalk/simpsidewalk
  (pop (getr OBJECT)))

```


APPENDIX B

Scene Model Data Structures

```
entityp = ^entity;
entitylistp = ^entitylist;
entitynodep = ^entitynode;

entity = record
    id: integer;
    name: TokenString: (name of an object)
    attr: tattrlistp: (pointer to the list of attributes associated
                      with an object)
    faces: facelistp: (pointer to a facelist)
end;

entitylist = record
    first,last:entitynodep
end;

entitynode = record
    e: entityp;
    next:entitynodep
end;

entitrec = record (used for iterating through the entity list)
    flag:packed array [1..NFLAGS] of boolean;
    en:entitynodep;
end; (entitrec)

EntityRel = packed array [1..MAX_ENTS_WORLD,1..MAX_ENTS_WORLD] of
    boolean;
```

References

- [Bates 81] Bates, Madeleine. The Theory and Practice of Augmented Transition Network Grammars. In Leonard Bolc (editor), *Natural Language Communication with Computers*. Springer-Verlag, 1981.
- [Glicksman 83] Glicksman, Jay. Using Multiple Information Sources in a Computational Vision System. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence*. 1983.
- [Herman 83] Herman, Martin, Takeo Kanade, Shigeru Kuroe. The 3D MOSAIC Scene Understanding System. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence*. 1983.
- [Herskovits 82] Herskovits, Annette. *Space and the Prepositions in English: Regularities and Irregularities in a Complex Domain*. PhD thesis, Department of Linguistics, Stanford University, 1982.
- [Herskovits 84] Herskovits, Annette. *Space and the Prepositions in English: Regularities and Irregularities in a Complex Domain*. 1984. Draft: University of California, Berkeley.
- [Hochberg 71] Hochberg, Julian. Perception: 1. Color and Shape. In Kling, J.W., L.A. Riggs (editor), *Experimental Psychology*. Holt, Rinehart and Winston, Inc., 1971.
- [Hopcroft and Ullman 79]
Hopcroft, John and Jeffrey Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Company, 1979.
- [Hwang 83] Hwang, Vincent, Takashi Matsuyama, Larry Davis, Azriel Rosenfeld. *Evidence Accumulation for Spatial Reasoning in Aerial Image Understanding*. Technical Report, Center for Automation Research, University of Maryland, October, 1983.
- [Johansen, et al 83]
Johansen, P., N. Jones, J. Clausen. A Method for Detecting Structure in Polyhedra. 1983. Institute of Datalogy.
- [Korein 84] Korein, James. *A Geometric Investigation of Reach*. PhD thesis, Department of Computer and Information Science, University of Pennsylvania, 1984.
- [Krotkov 84] Krotkov, Eric. *Construction of a Three Dimensional Surface Model*. Technical Report, GRASP LAB, CIS Department, University of Pennsylvania, 1984.
- [Potmesil 83] Potmesil, Michael. Generating Models of Solid Objects by Matching 3D Surface Segments. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence*. 1983.
- [Radack, et al 84]
Radack, Korein, Ganis, McNally, Korein, Shapiro. *NASA Programmer's Guide* CIS Department, University of Pennsylvania, 1984.

[Reynolds, et al 84]

Reynolds, G, N.Irwin, A.Hanson, E.Riseman. Hierarchical Knowledge-Directed Object Extraxction Using a Combined Region and Line Representation. In *Vision Workshop*. 1984.

[Rosenthal 81] Rosenthal, David. *An Inquiry Driven Vision System Based on Visual and Conceptual Hierarchies*. UMI Research Press, 1981.

[Shapiro 84] Shapiro and Haralick. A Heirarchical Relational Model for Automated Inspection Tasks. In *Int. Conf. on Robotics, Atlanta, Ga.*. 1984.

[Talmy 83] Talmy, Leonard. *How Language Structures Space*. Technical Report 4, Berkeley Cognitive Science Report, January, 1983.

[Tropf 83] Tropf and Walters. An ATN for 3-D Recognition of Solids in Single Images. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence*. 1983.

[Winograd 83] Winograd, Terry. *Language as a Cognitive Process*. Addison-Wesley Publishing Co., 1983.

[Winston 79] Winston, Patrick Henry. *Artificial Intelligence*. Addison-Wesley Publishing Company, 1979.

[Winston and Horn 81]

Winston, Patrick and Berthold Klaus Paul Horn. *LISP*. Addison-Wesley Publishing Company, 1981.