TOWARDS AN INTELLIGENT COMPUTER GRAPHICS SYSTEM

Marek Holynski, Brian R. Gardner, Rafail Ostrovsky
Boston University

BUCS Tech Report # 86-003
January 1986

Computer Science Department
College of Liberal Arts
BOSTON UNIVERSITY
Boston, MA 02215

TOWARDS AN INTELLIGENT COMPUTER GRAPHICS SYSTEM

Marek Holynski
Computer Science Department
Boston University
Boston, MA 02215
and
Center for Advanced Visual Studies
Massachusetts Institute of Technology
Cambridge, MA 02139

Brian R. Gardner, Rafail Ostrovsky
Computer Science Department
Boston University
Boston, MA 02215

Abstract

The development of an interactive computer graphics system that ties the meaning of a picture to its graphic representation is discussed. The system utilizes a technique for knowledge representation which is relevant both for computer graphics and for artificial intelligence. The description of relations among picture elements and concepts that are represented by these elements is provided in the form of a semantic network and expressed in Lisp. In order to display knowledge described by semantic networks, a Lisp graphics package is used. By integrating semantic network and Lisp graphics package we are able to analyze, create, and modify graphics from the standpoint of contextual understanding.

1. Introduction

Given data can be represented on a computer screen in an almost infinite number of ways. Only a few of the resulting pictures, however, can optimally or even suitably capture a specific meaning within the desired or necessary criteria for a particular user. Selection of the appropriate representation from a large number of all possible representations can be a formidable task for human viewers. Existing computer graphics systems do not provide any assistance in easing this task and require the user to take full responsibility for object formation and picture composition. The situation would improve when, instead of burdensome testing of different versions of an image, the user could obtain some support in deciding about graphic presentation from a knowledge-based computer graphics system.

Knowledge-based graphics systems tie the meaning of a picture to its graphic repre tation. Therefore such a system should consist of equally powerful tools for both knowl and graphics representation. Existing systems have not addressed the problem of con tually integrated representation. One obstacle is the lack of relevant theory incorpora precise mathematical descriptions of visual criteria in a formal way. Another obst arises from past approaches of system designers. Their attempt to combine existing and graphics packages without modifications led to communication problems between two components. The transformation of a representation schema suitable for AI ap cations into the schema most efficient for graphics purposes proved to be inefficient difficult. The crude interface between these different methodologies, often merging different levels of language (assembler-written graphics subroutines and Lisp), resulted systems with severely limited capabilities.

We propose to solve this problem through a new approach. We utilize a technique knowledge representation which is relevant both for computer graphics and for artifi intelligence. The description of relations among picture elements and concepts that represented by these elements are provided in the form of a semantic network. The seman network processing system (SNePS) [1] we use is written in Lisp. The system is capa of including both factual assertions and rules of inference in the same description. T capability allows references to indirectly specified picture elements as well as learning abo different beliefs and preferences of individual users and classes of users.

In order to display knowledge described by semantic networks, a Lisp graphics packa (Graflisp) [2] was used. Graflisp can perform tasks like graphics pattern matching a creating graphics primitives with list processing. It offers an object hierarchical netwo allowing manipulation of complex relationships between objects and sub-objects. Grafli has the potential to interface with existing AI packages, runs on a wide range of machin and can use any input/output device.

By integrating SNePS with Graflisp we are able to analyze, create, and modify graphi from the standpoint of contextual understanding. Our system also will be capable discovering the significant levels of picture attributes and select the relevant ones. T attributes used include number, size, position of picture elements and other factors su as rotational or perspective transformations, color, complexity, variety, and regularit Our knowledge-based graphics system uses selected values of picture attributes in order generate, change and refine images interactively. The presented system is relevant both computer graphics and to artificial intelligence. It can be used as a stand alone tool and the future can serve as the backbone for knowledge-based. intelligent graphics application

2. Semantic Networks

Semantic networks are commonly used as one of the knowledge representation formalisms within the AI community. However, there are drastic differences between different approaches taken. The only common ground from system to system is the existence of some graph of "nodes" connected by "links" (which sometimes are called arcs). Every system attributes its own semantics to nodes and links. The only reason why these networks are referred to as "semantic" is because they usually try to represent the *meaning* of some "objects". Since networks are just syntactic structures, network operators have to specify what is the semantic meaning of the structure. However, what operations and objects are and what they represent is different from system to system.

Usually nodes represent some concepts, objects or events and links represent there interrelations. Originally, nets have been developed as a psychological model of human associative memory. However, in many, if not all, psychology-based networks, the mapping between logic-based representation and semantic net representation is absent. Furthermore, many such systems do not represent quantified rules within a semantic network. We need a system which is capable of representing inference rules and which is able to perform both forward and backward chaining. In addition, we don't want to allow nodes to represent any objects, events or states, but rather *concepts* about events, objects and states.

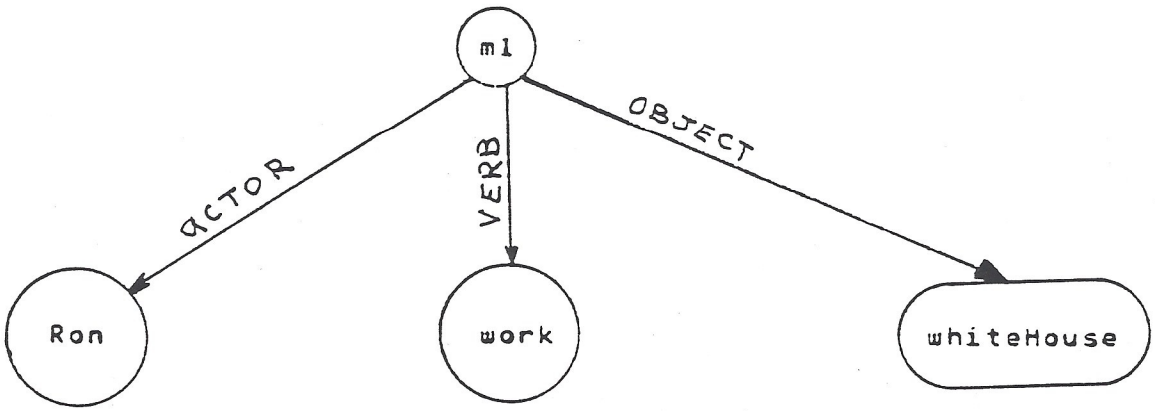


Figure 1. Semantic network representation of a single statement

We are going to deal with a particular semantic network which we feel is most suitable for our needs. For building and operating on this network we utilize SNePS system. In SNePS knowledge is represented as a labeled directed graph in which every node represents a concept. Edges between nodes demonstrate relations among these concepts. Thus, for

any new concept a new node must be created. For example if we want to represent a statement p(a,b) where p is some relation, we will have to have at least three nodes: one for a, one for b and one for a concept representing a proposition that p(a,b). Fig.1. illustrates this idea. Node m1 represents entire proposition: "Ron works in a White House". Now, we can reason about a particular concept p(a,b). This is different from other systems, where user is allowed to create a direct link between a and b and call it p. In SNePS it is not allowed to have a node Ron connected to a node whiteHouse by the link "work".

If two nodes have an edge going between them, then they are conceptually related and their *meaning* is defined in terms of each other. In general, the meaning of every node is embedded in the entire structure of the network. We are capable of defining more complex concepts in terms of other nodes of our semantic network. This capability makes SNePS particularly beneficial for representation of graphics-related concepts, since most of the picture requirements and user preferences require an abstract level of specification. The knowledge about these higher level constructs can be built incrementally, using more primitive concepts as a building blocks.

SNePS is capable of representing both factual assertions and rules of inference in the same network. The SNePS inference mechanism allows us to use both forward and backward chaining rules as well as by-directional reasoning. All logical connectives and quantifiers are directly expressible in the system. Negation is included as well. Thus, both classical negation and negation by failure can be reasoned about. SNePS enables us to represent different beliefs and preferences of different users or users' groups. For example, we can represent statements as following: "Architects prefer people to be displayed to the right of the house." Semantic network representation of this statement is shown in Fig.2.

The system allows us to have variables which refer to single nodes as well as to all parts of the network, enabling us to have meta-level reasoning. Using this, we can create an intelligent graphics system which will adjust to the preferences of different users, and have a whole framework of meta-level rules about applicability of various object-level rules in different environments.

During deduction a special purpose unification algorithm looks for unifiable instances of the parts of the semantic network against parts of the rules. Sometimes, however, instead of the unification we desire some computation to be performed. Thus *procedural attachment* is an important feature of the deduction system. For this purpose special *function nodes* are incorporated into SNePS' inference mechanism. The idea is that, rather then doing unification, a function node will execute some Lisp code associated with it and than pass success or failure to a deduction system. The availability of such nodes will allow us to reason about Lisp functions as conceptual entities. Thus, we will be able to reason about Graflisp routines as function nodes at the SNePS inference level.

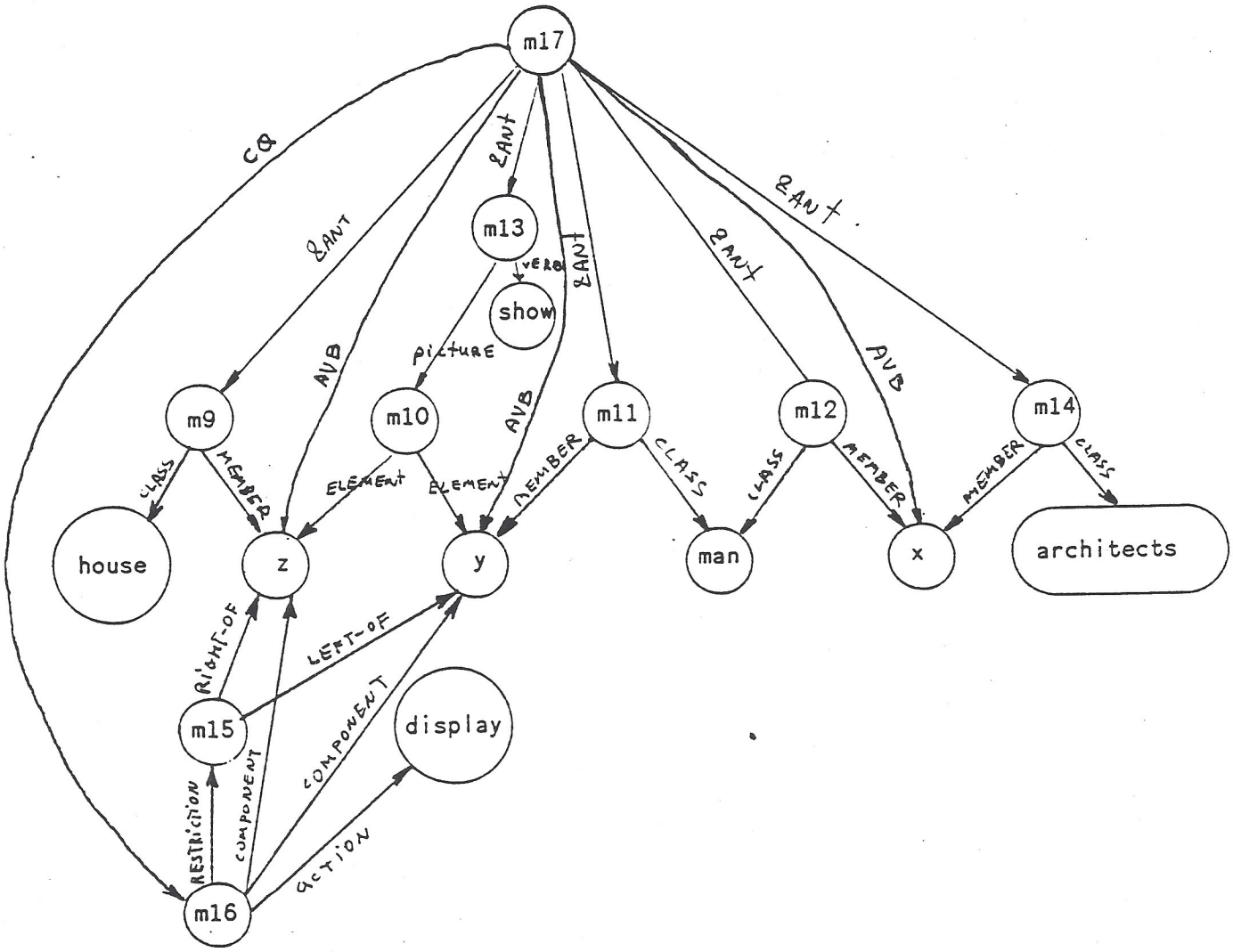


Figure 2. Representation of users' preferences in a semantic network

We can view representation of all nodes in a semantic network as being intensional concepts. We will view Graflisp objects as being extensional. SNePS intensional concepts will be connected with their extensions. This will allow us to represent such notions as a "house" and a "home" as two different concepts in a semantic network, which refer to the same graphical object (i.e. Graflisp picture).

3. Graflisp

Graflisp is the graphics package specifically designed for Artificial Intelligence applications (its first version was completed in 1982). Unlike standard graphics packages, Graflisp incorporates many AI techniques into its advanced graphic manipulations, such as a pattern matcher for graphical properties, color name specifications, hooks for intelligent blackboard monitors and error handling, and use of self-modifying code. Written entirely in Lisp, Graflisp readily interfaces with currently existing AI programs.

The overall theme used in writing Graflisp is to offer a package of intelligent graphics functions for the needs of the AI community. Graflisp uses a hierarchical lattice based object network to represent complex object relationships, known as a *parts hierarchy*. This allows relevant knowledge about parent objects, such as size and orientation, to be communicated to its sub-objects via heredity at render-time.

Objects can be flexibly designed in a variety of ways. Graflisp allows objects to be *data defined* (the standard method) and/or *procedurally defined*. Data definitions generally involve describing objects in terms of lists of data points, such as building an object's surface up from a finite number of polygons. A procedural definition allows objects to be defined in terms of functions which procedurally define an object by its type. In addition to the procedural definitions available in the package, such as spheres, lights, and surfaces of revolution, Graflisp also offers functions for the interfacing of specialized user-written object types.

Graflisp is hardware independent. It currently operates on a variety of machines (VAX 11/780, VAX 11/750, IBM 3081, etc.), operating systems (UNIX, VMS, and VPS), and Lisps (Franz Lisp, BU-Lisp, and is portable to other Lisps). It is also *device independent*; that is, it can be used on a wide variety of graphics devices, such as an AED-512, VT125, Tektronix 4013, and Ramtek color terminals. Graflisp uses logical devices which, in addition to being graphics hardware, can be pipes to other software packages. Functions also exist for interfacing user written device drivers, allowing a programmer to interface to virtually any piece of hardware or software.

Several applications have begun to explore the potential of Graflisp: a simulated model of a robot arm, a public information system, and a handwriting generator. The robotic arm utilized Graflisp's object hierarchy to model the relationship between the arm's components. This allowed easy transformations of the relative orientations of the various parts of the arm. The public information system was designed as a service for tourists. It interacts with users by discovering their preferences for tourist sites and, after analyzing them, recommends which of Boston's sites would be most interesting. Then it displays a map of Boston's public transportation system and illustrates the best route from the visitor's current location to the site.

The handwriting program is a turnabout of the standard AI character recognition problem. Given a typed file, it produces handwritten output (Fig.3). Rather than using digitized images of letters, causing a mechanical looking script, this program uses information about the vector forces of the pull of a particular individual's hand exerted on a pen in order to characterize that person's handwriting style. Using Graflisp functions, the program can transform style information according to the speed, size, and position of the output, while including a proportional randomness. A continuous stream of hand movements is calculated as patterns in the selected handwriting style and matched to patterns in the input text. Output generation uses Graflisp's B-spline curve function. Pull vectors are related to control points which alter the curvature factor to reflect writing speed and quality. In many cases the results are indistinguishable from the handwriting of the corresponding human author.

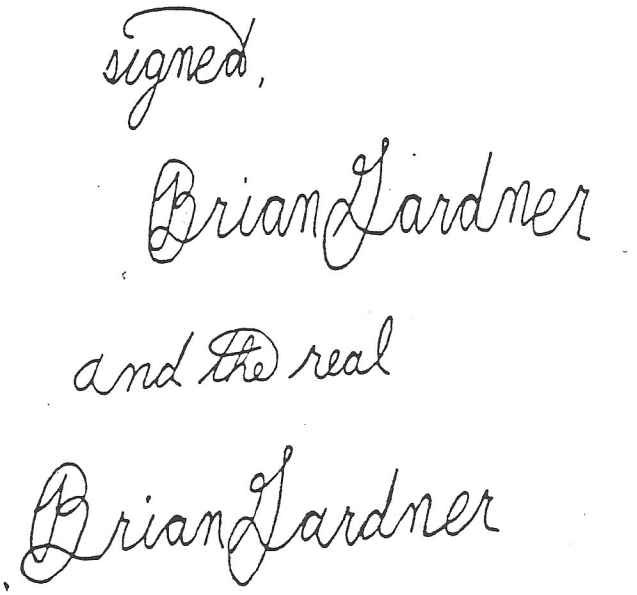


Figure 3. Handwritten output produced by Graflisp (top) and by the corresponding human author (bottom)

These simple examples are evidence to Graflisp's potential. For our purposes, Graflisp's internal network structure makes it particularly amendable to combination with SNePS. By utilizing both of these powerful tools we are creating a system that can associate abstract meanings and relationships with relationships between objects in a picture hierarchy. Such associated relationships are necessary for expert systems dealing with human visual perception and cognition. Using human-like reasoning, our system will be able to create a visual representation appropriate for a given subject or audience, and change that image according to current needs.

4. Reasoning About the Picture

To test our approach we decided to explore a domain in which picture-generation has to be derived starting from the conceptual level. We have chosen a real life situation, simple enough to start with, but complex enough to verify our methodology. The presented example illustrates a case for reasoning about a group of people and their locations. The goal was to make the system accept abstract requests and decide upon a *conceptual scene* (a set of conceptual components and their interrelations which corresponds to a given request) as well as select *actual picture* specifications.

We started with concepts like: people, houses, people living in particular houses and working at particular places. The facts that were given to the system are:

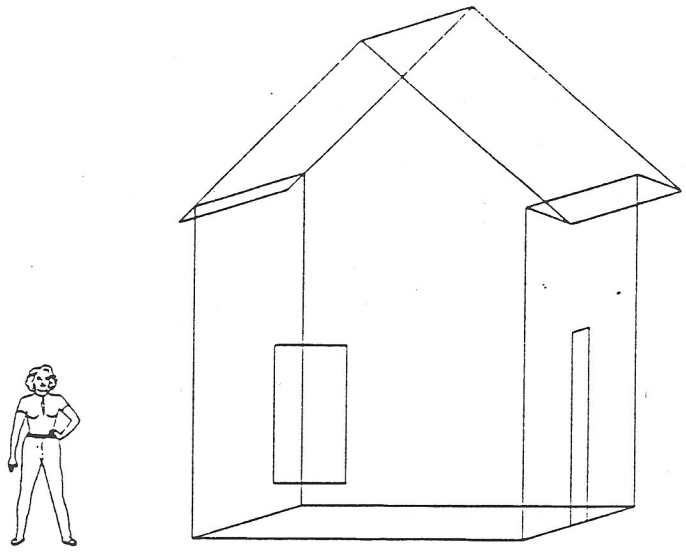
- `Ron, Bill and Jane are people.`
- `Ron works and lives in a house by the name ''White House''.`
- `Bill works at ''White House'' but lives in the ''Regular House''.`
- `Jane both works and lives in the ''Regular House'' with Bill.`

Furthermore, we tested temporal reasoning capabilities, using queries about location of people with respect to their houses or workplaces and current time of the day. For example, we could provide such a rule to a system as "People are at their workplace during work-day and at home otherwise", and let the system select a conceptual scene according to the present time. The system would believe now (system beliefs, of course, don't have to be always true) that people work during working hours and are at home when not at work. When we ask the system to show a home of Bill, the system should first check the time and realize that the home of Bill is a particular house. If it is working hours, it will show Jane and the Regular House as in Fig.4a (it will not display Bill, since he is at work). During a week-end or not-working hours, the system will display both Jane and Bill and the Regular House (Fig.4b). *

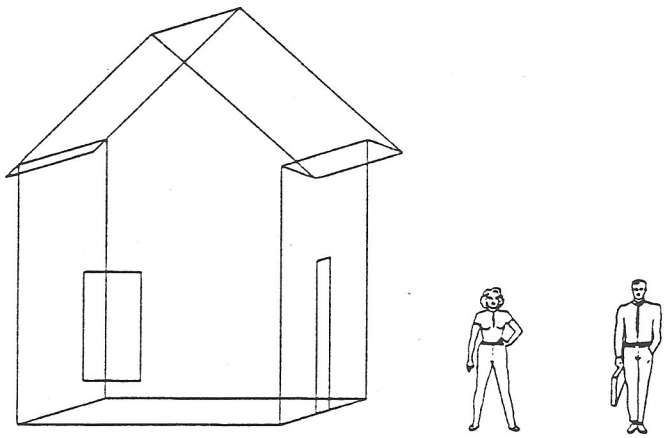
The system is capable of representing abstract concepts. When we ask the system to display such an abstract concept, the system tries to find what this abstract concept means and what it should be tied to. A home or a workplace are examples of such abstract concepts. If we request the system to show a workplace of somebody, the system would have to realize that a workplace is some particular building where the referenced person works. It then would have to find out what that building is, check the time, and display this building together with the appropriate people. For example, if we ask for the workplace of Ron during working hours, the system will display the White House, and both Ron and Bill. The same request during off-hours will produce only Ron and the White House.

---

* Pictures produced by Graflisp are representing solid objects in colors. To increase readability and only for purpose of this publication, we use wireframe representation of these objects.



a) Picture generated at 11am



b) Picture generated at 7pm

Figure 4.

The system decides on the appropriate scaling of the picture elements. It, for instance, will scale down people with respect to the White House and place them in a default location. Furthermore, more complex decisions on positioning of picture elements in the picture can be made. For example, we told the system that we like people to be displayed in front of the White House, but to the left of the Regular House. During construction of the picture the system takes user preferences into account.

5. Picture Composition

Implementation of the simple example from the previous section allowed us to test the abilities of SNePS to describe input information and reason about a picture as a relevant semantic network, Graflisp's capacity of representing this description as an image, and the efficiency of the interface between these tools. To input facts given as four statements about three people and their locations we used SNePS User Language, SNePSUL. SNePSUL is a Lisp based interactive environment consisting of a set of functions among which "build" is the main mechanism for adding new information to the network.

To represent the statement: "Ron, Bill and Jane are people" as the network in Figure 5, we have to input the following SNePSUL functions:

```
(build member Ron
       class person)
(build member Bill
       class person)
(build member Jane
       class person)
```

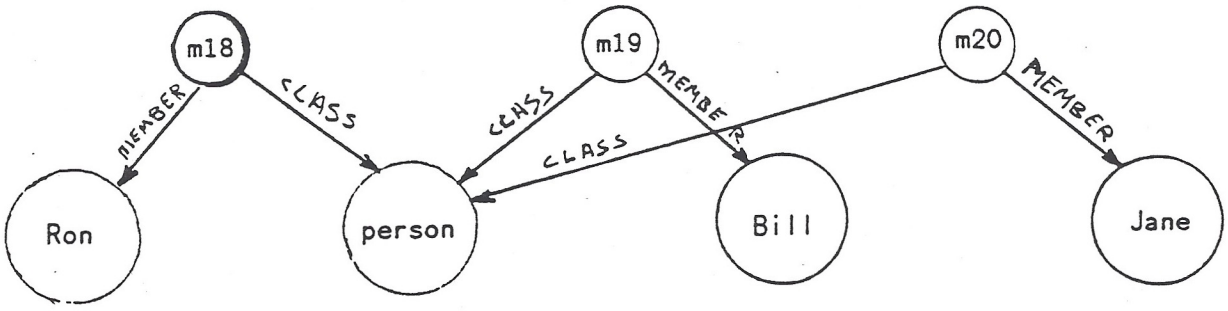


Figure 5. The network representing the statement "Ron, Bill and Jane are people".

To represent the statement: "Ron lives and works in a house by the name "White House"", as the network in Figure 6 we enter:

```
(build verb live
       actor Ron
       place house
       placeName whiteHouse)
(build verb work
       actor Ron
       place house
       placeName whiteHouse)
```

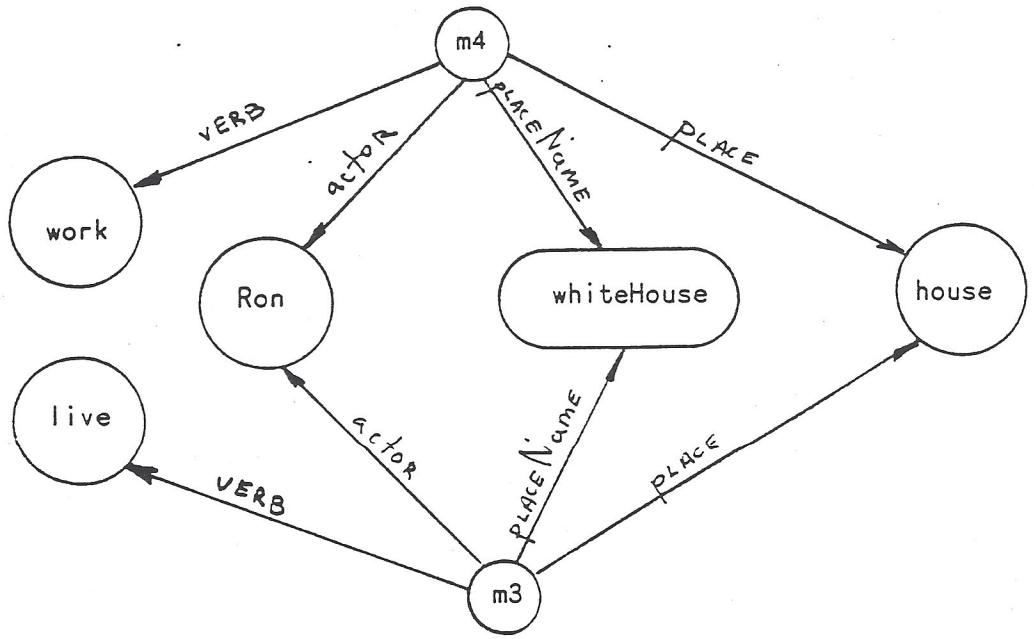


Figure 6. The network representing the statement "Ron lives and works in a house by the name White House."

Every concept is represented by a unique node. Thus, nodes may be shared in different propositions. For example, "Ron", "house" and "whiteHouse" nodes are shared in the network presented in Fig. 6.

In a similar fashion we can combine facts from statements describing living and working places of people in our example (Fig. 7).

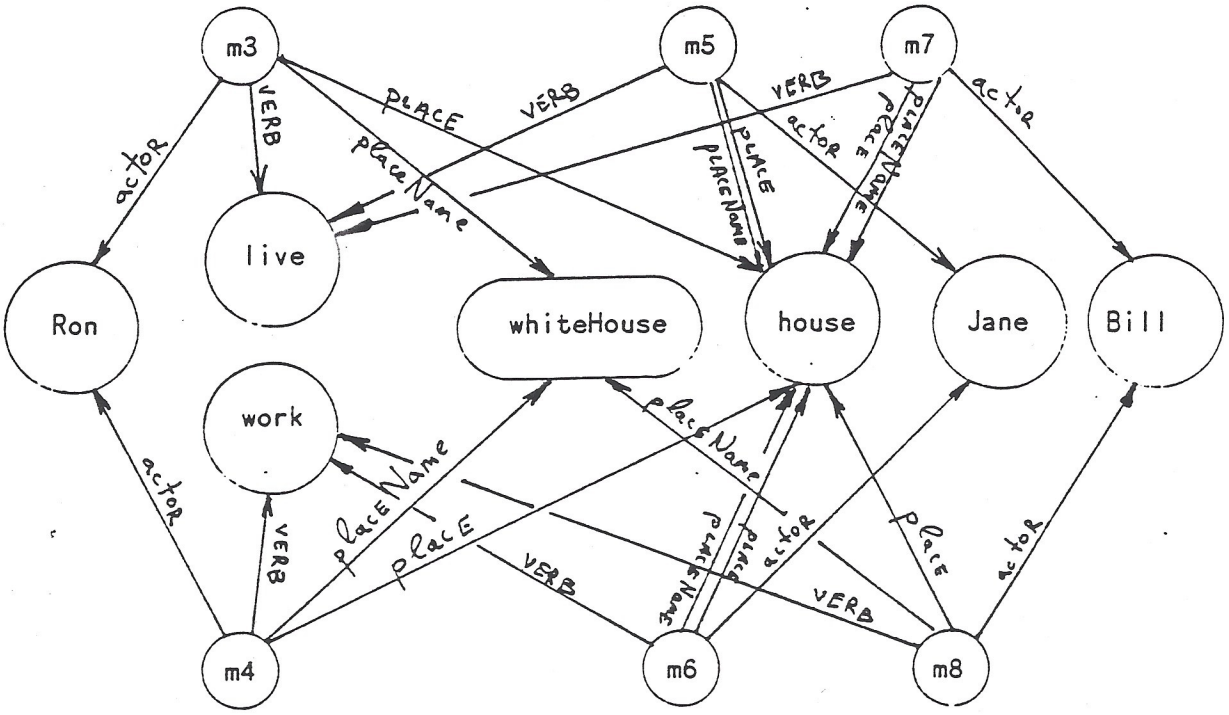


Figure 7. The network representing living and working places of Ron, Bill, and Jane

Concepts that can be represented as picture elements (objects) are known to the system. These objects are created from an initialization file. An example below shows an initialization file which contains the Graflisp code needed to generate commonly used object - a house from Fig.4 - and set its respective properties, such as height, width, depth, class, type, and color.

```
(create 'hbody '((5 9.5 1) (2 6.5 1) (2 0.5 1) (8 0.5 1) (8 6.5 1)
        (5 9.5 1))) ; create side view of house
(recreate 'hbody (pipette hbody '(0 0 20.0))) ; extrapolate it into 3-D
(putpic 'hbody 'obtype 'solidpolygon)
(putpic 'hbody 'color 'blue)
(create 'roof '((5 9.5 0.9) (5 9.7 0.9) (2 6.7 0.9) (2 6.5 0.9) (5 9.5 0.9)))
(recreate 'roof (pipette (list roof
```

```
                (reverse (rotate roof '(5 9.6 0.9) 0 (pi) 0)))
              '(0 0 20.2)))
(putpic 'roof 'color '(0.35 0.35 0.35))
(create 'door '((8.03 0.5 9.5) (8.03 4 9.5) (8.03 4 12.5)
        (8.03 0.5 12.5) (8.03 0.5 9.5))) ; create face of door
(recreate 'door (pipette door '(0.1 0 0))) ; extrapolate door into 3-D
(putpic '(door roof) 'obtype 'solidpolygon)
(putpic 'door 'color 'maroon)
(create 'window1 (movegraf (box '(3.5 4.8 1.5 4.0)) 0 0 0.95))
(recreate 'window1 (pipette window1 '(0 0 0.1)))
(putpic 'window1 'obtype 'solidpolygon)
(putpic 'window1 'color '(.4 .4 .41))
(create 'house)
(subpic 'house '(hbody door window1 roof))
(rotatpic 'house '(5 5 11) '(0 0.1 0))
(scalepic 'house '(0 0 0) '(2.0 2.0 2.0))
(putpic 'house 'height 20.0)
(putpic 'house 'depth 40.0)
(putpic 'house 'width 20.0)
(putpic 'house 'class 'building)
```

In addition to objects created in the initialization file, objects can be added interactively. Much of this is made more efficient by Graflisp's graphical extrapolation functions and procedurally defined object types. Objects' definitions for our system usually take on the appearance of Lisp code, rather than files of coordinates.

The simplest form of data is comprised of list structures. It is natural to think of a point as a list of (x y) or (x y z) coordinates, a polygonal line as a list of points, and faceted shapes as a list of polygonal lines. Data for non-polygonal structures is usually made up of points, lists of points, and lists of scalar parameters. Our system can accept the raw data specifications for an object by simply passing it to the "create" function, as was done in creating "hbody" in the initialization file. However, often the raw data for an object is either unmanageably large, unavailable, or could be simplified by parameterization. Here, we have taken advantage of Graflisp's functions to parameterize our data whenever possible and approximate an object's shape when its exact raw data is not available.

Much of the ease of object creation is due to functions which generate data. These functions are called *extrapolation functions*, because they extrapolate small amounts of data into much larger ones. For example, the body of the house is created by first defining one end of the house with raw data; this one wall is then passed to the "pipette" function,

which extrapolates it into a three dimensional collection of polygons forming a house shape twenty units deep. The "create" function then takes the resulting data and assigns it to an object named "hbody", a name which can be later referenced in the initialization file to set this object's properties and orientation. "Pipette" is a particularly useful function, since it can often extrapolate a cross-section of a building into an entire architectural structure.

Another useful tool in object creation is the "obtype" property of Graflisp objects. This property allows a wider choice of building blocks with which to design objects. While most graphics packages limit the designer to using either polygons or spheres to form objects, our objects can be comprised of both. Since the data formats are compatible, we may even change an object's type on the fly (this is very useful during previewing). Additionally, Graflisp offers other object types. Particularly useful is "yrevolve" which generates surfaces of revolution from only the outline of the shape. Using these "obtypes" in combination allows complex objects to either be replicated exactly, or represented in an approximated form via parameterized data constructs.

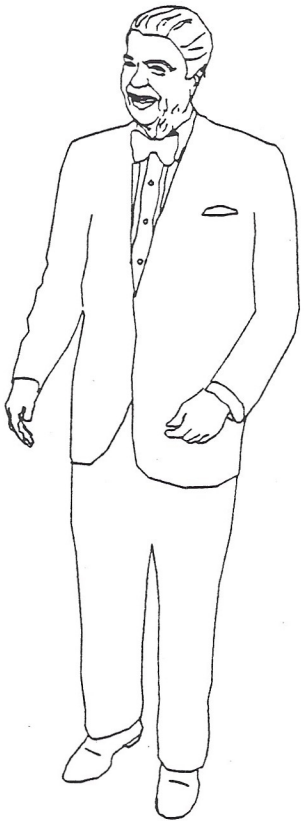


Figure 8. Wireframe version of Ron; in the object type representation Ron is comprised of spheres, polygons, and surfaces of revolution

Perhaps the most important feature of Grafisp is its ability to maintain clear relationships between object and sub-object. By using the "subpic" function, we can organize our objects around a hierarchical lattice structure. During rendering, information such as orientation and color are passed from parent objects to child objects via inheritance. This allows us to design not only the physical parts of objects, but to model physical relationships between the parts as well. For example, "Ron's" hands are connected to his fore-arm at a fixed distance from his elbow. To bend Ron's arm at the elbow, it is only necessary to rotate his fore-arm about the elbow, and the fore-arm's sub-objects, such as his hand, will rotate with the fore-arm automatically. Additionally, since each object can be referenced relative to its own coordinate system, the bending of any sub-object about a joint is accomplished by simply rotating it about its origin. If reference points other than the object's own origin are needed, they are incorporated into the object's property list with an appropriate name and value.

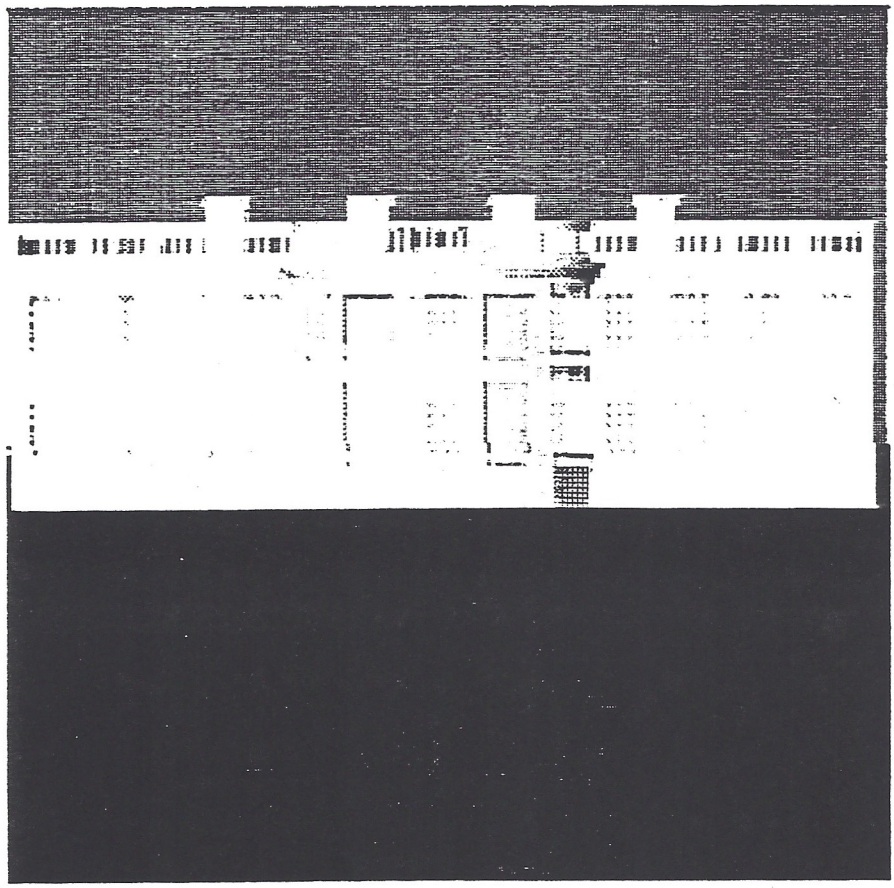


Figure 9. The White House uses subpicturing of simple pipettes to achieve a complex structure

Subpicturing is also useful in designing architectural structures. For example, the entire structure of the White House in Figure 9 is comprised of less than a dozen base

objects. The base objects are formed by data generated from the "piperte", "poly", and "box" extrapolation functions: hence, no more than five parameters were ever needed to create any visible object. Most of the structure is built using subpicturing. For example, although there are twenty-six windows in the image, only the data for two windows must be specified (the top window and the bottom window). Since the White House is predominantly comprised of panels, we can define it in terms of a dataless parent object called "whiteHouse" with eight dataless sub-objects labeled "whpanel1" to "whpanel8" which define the architectural relationships forming the front wings of the White House. Similarly, the White House's front porch can also be defined as a dataless object which as five of these panels labeled "whfpanel1" to "whfpanel5" (see Figure 10). Each of these panels has the same data object, labeled "whfpanel", subpictured to it, defining the visual appearance of a panel. This base panel has some sub-objects of its own, such as a triangular window ornament and the upper and lower windows. A similar hierarchical structure is used to form the slightly different looking relational structure of the panels on the porch section, which uses many of the same base data objects.

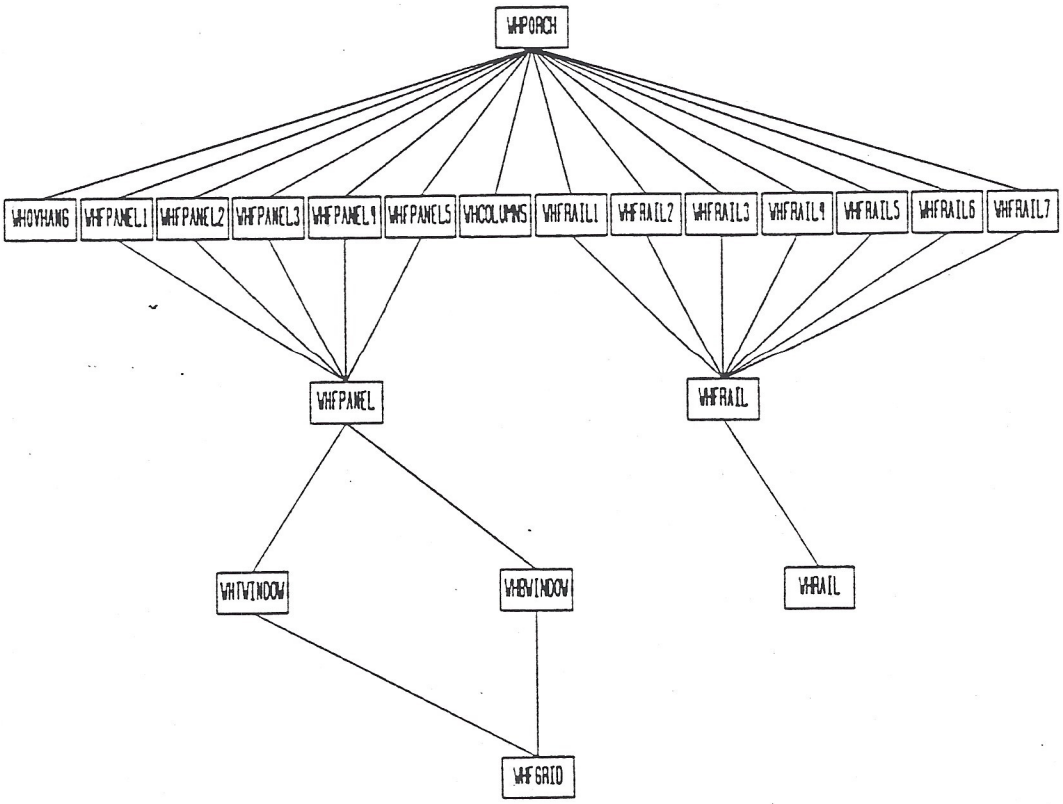


Figure 10. Architectural relationships of the White House porch represented in an object hierarchy

6. Inference Reasoning for Scene Composition

At the inference level, the system has to have a way to assume that some picture element has been displayed. Function nodes give a procedural attachment capability to the otherwise declarative style of SNePS programming. To "prove" a function node, the system must call a function which is associated with it. This is the actual SNePS-Graflisp interface. Here is one example of such a display rule:

```
; For all x, y, and z
;   if x is living in a place z by the name y
;     then if process of drawing y succeeds
;       it must be the case that we displayed it.
(build avb ($x $y $z)
        ant (build verb live
                   actor *x
                   place *z
                   placeName *y)
        cq (build ant (build name:  showPicture
                              placeName *y)
                  cq  (build action display
                              description (build verb live
                                                 actor *x
                                                 place *z
                                                 placeName *y)
                              type *z)))
```

This rule allows the system to reason about a particular action: displaying the living-place of one of the people known to the system. The rule itself is presented first in English version (lines starting with semicolons) and then as an actual SNePS notation. Entering the above SNePSUL commands will create the network in Figure 11.

In the above rule "avb" arc stands for "all-variables-bound" and represents a universal quantifier. SNePS uses its rules in both backward and forward reasoning. The way this rule can be interpreted during forward reasoning is: if antecedent m21 is true then a consequent m25 is true. The consequent is also a rule. The antecedent, labelled m24, of this embeded rule is:

```
(build name:  showPicture
              placeName *y)
```

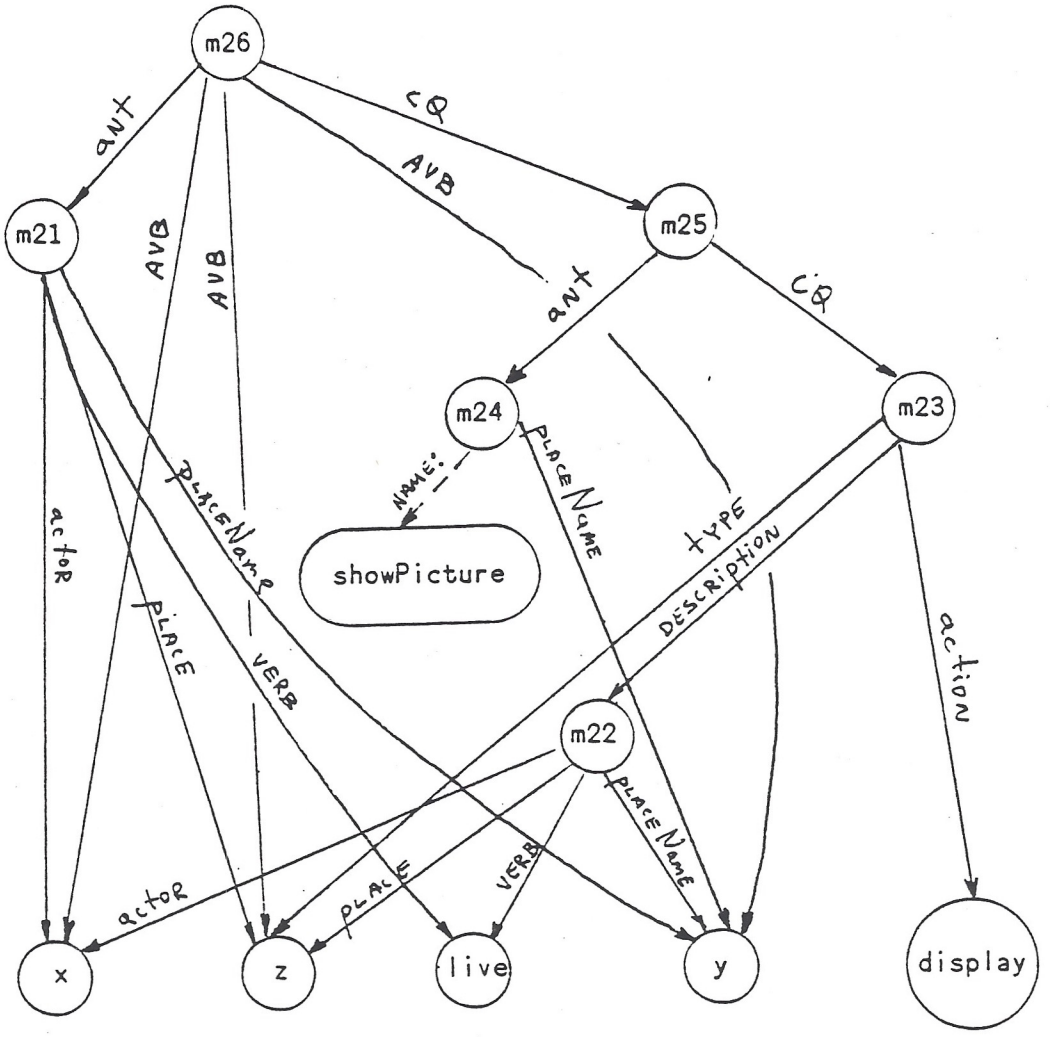


Figure 11. Semantic network representation of a display rule

The arc named "name:" is a special system predefined arc which points to a function node. In diagrams of the semantic network it is drawn as a broken line. A *function node* creates a process which calls a Lisp function with an argument bound to y; this process can either succeed or fail. If the above process succeeds, then the consequent m23 of the rule is asserted.

If we had used this rule in a backward chaining, we would have tried to prove that for some actor x, living somewhere at place z with the "placeName" y the picture had been displayed. If none of the variables were bound, this would be similar to the query: "Show me all the places of living for all the people". If, on the other hand, some of the

variables were bound, it would be a reference to some specific instance of x or y or z. For example, if x is bound to Ron and y and z are free, this would be equivalent to the query: "Show me the place in which Ron lives". To prove this, we would have to prove that the entire nested rule is valid. Thus, an instance of m21 would have to be found with some unifiable binding for x, y and z. If such an instance was found in our graph, the nested rule m25 would then be executed. This rule would create a SNePS process, which would call Graflisp. When Graflisp displays a picture, the function node succeeds and returns true, and then the final consequent would be asserted.

A similar rule can be created for displaying a person's workplace. The network representing this rule will be almost identical to the network presented in Fig. 11. The only difference will be that the verb "live" will be replaced by the verb "work".

The next step is to create rules which will deduce what a home and workplace is in terms of a house.

```
; For all x
;   if x is a person
;     then for all y
;       if y is a house where x lives
;         then y is a home of x.
(build avb $x
       ant (build member *x
                  class  person)
       cq  (build avb $y
                  ant (build verb live
                            actor *x
                            place house
                            placeName *y)
                  cq  (build verb live
                            actor *x
                            place home
                            placeName *y)))
```

Similar rule is created to represent a rule that a place where person works is a workplace. It looks just like the above rule with "home" substituted for "workplace" and verb "live" changed to verb "work". Now, we can express the rule that all people are at work during work-hours and at home during off hours:

```
; For all x
;   if x is a person
;     then if currentState is workTime
;       then x is at his workplace.
(build avb  $x
       ant (build member *x
                  class person)
       cq  (build ant (build currentState workTime)
                  cq  (build verb currentlyPresent
                             actor *x
                             place workplace)))
```

A similar rule is given to the system to deduce: if it is a "freeTime", then a person must be at home. The system may not know what a home (or a workplace) of x is. It just assumes that an actor is at some workplace or home. Then, it will have to deduce what is a particular home or workplace of x. We will show a rule which infers that a person is at home. Similar rule is created for a workplace.

```
; For all (Actor, place, placeName)
;   if  Actor lives in a place by the name placeName
;     and
;   Actor is currently present at his home
;     then home name is placeName.
(build avb ($actor $place $placeName)
       &ant (build verb live
                  actor *actor
                  place *place
                  placeName *placeName)
       &ant (build verb currentlyPresent
                  actor *actor
                  place home)
       cq   (build verb locatedAt
                  actor *actor
                  place home
                  placeName *placeName))
```

But how does the system know whether it is a workTime or a freeTime? To figure that out the next set of rules is built:

```
(build avb ($day $hour $minute)
       ant (build name:  isItWorkTime
                  dayOfTheWeek *day
                  hour  *hour
                  minute *minute)
       cq (build currentState workTime))
(build avb ($day $hour $minute)
       ant (build name:  isItFreeTime
                  dayOfTheWeek *day
                  hour  *hour
                  minute *minute)
       cq (build currentState freeTime))
```

These two rules assert that it is a workTime/freeTime if and only if a function node isItWorktime/isItFreeTime succeeds. We use a non-monotonic reasoning, so the state may change from workTime to freeTime. The system must not use old assertions about the time; instead, it must check the time again if it needs to. The way to solve this problem is to remove the temporal assertions each time, thus forcing the system to deduce them again.

When SNePS needs to display a set of objects, it creates a process which queries Graflisp about its capability to display those given objects. If Graflisp is capable of displaying the given set of objects, it collects, orders, orients, composes and renders that given set into an image. Then Graflisp passes a success message to the SNePS process, which consequently enables SNePS to deduce that the conceptual request is displayable. If the description which the SNePS was provided with on a conceptual level can not be visualized in an image, Graflisp will be unable to find the corresponding objects or rules necessary to compose the scene and will pass failure back to the SNePS process.

Based on the information being passed from SNePS, the Graflisp module of the system is responsible for composing and rendering the image within the constraints of its view camera. The view camera determines the area of space to be viewed, the degree of perspective deformation, and the orientation of the image it will produce. This is analogous to a camera in the real world.

Scene composition is accomplished in much the same way a photographer might compose a scene of some miniature models on a table top. First, the camera is placed at a position that would be roughly eye-level relative to the objects being viewed. Then a rectangular plane is positioned in the scene, large enough to serve as a flat, green, ground-like surface on which the objects are to be placed. Behind the far edge of this ground object is placed a vertical blue backdrop; this planar surface will represent the sky in the final

image. These three objects form the first hierarchical level of the objects to be subpictured to the "world" being composed.

Next, the objects are placed into the scene. They are first ordered in front of the camera from left to right so as to match the expectations of the viewer. In order to do this, the system utilizes production rules which relate the user requesting the depiction to their personal preferences on the way in which objects should be ordered into a scene. Thus, a user who prefers people on the left side of an architectural structure will get a different picture than one who prefers people to the right of such a structure, even though they may have each made an identical request to the system.

After completion of the horizontal composition of the scene in image space, the system then composes the depth aspect of the scene. Since each object in the scene may be placed at a multitude of distances from the camera and still project onto the same horizontal position in the image, it is necessary to determine which distance is optimal for each object. To accomplish this, perspective geometry is used to project the object's image area into a three dimensional pyramid which defines the region of space which the object may occupy. Similar to a photographer moving an object towards and away from the camera to find the distance at which the object appears the correct size in an image (that is, not so close that it blocks the other objects in the scene, yet not so far away that it is hard to see), geometric ratios are used to determine the distance at which an object's projected image size is maximized without occluding its neighboring objects.

Once the objects are entered into the scene hierarchy as sub-objects linked and oriented relative to the ground, the completed scene is passed to the "see" function for rendering. Here each data object is transformed according to the combined inherited transformations from its position in the object hierarchy. They are then processed by culling and z-buffering algorithms to remove hidden surfaces prior to being rendered into an image and subsequently displayed. As the final step in this process, a predicate value is passed back to SNePS to signal the success or failure of Graflisp to perform the requested depiction.

7. Conclusion

In the past, graphics systems have placed the tasks of scene composition and object formation heavily on the user. They have required the user to be both a graphics designer and a computer programmer. Even if the user possesses these skills, he spends hours of additional time constantly re-adjusting the image trying to get it to look correct. It would be ideal if the user could speak to the computer in meaningful, perhaps even abstract, terms and have the computer understand and compose the scene according to the meaning of the picture. This, however, begs the age-old question, "Can computers reason about

composition of original scenes, or use intelligence to create images?".

It is perhaps historically significant that our experiment has potently answered this question with a "Yes". This experiment, however, was intended to be the first step in the creation of a graphics expert system. Expert systems are increasingly dependent on visual feedback, iconic input, and pictorial representation of results. Knowledge-based graphics can be highly desirable for this purpose. Since human experts analyze problems and illustrate solutions with pictures, we have come to expect computerized systems to do so as well.

The presented system gives us an opportunity to analyze, create, and modify computer images from the standpoint of contextual understanding by relating the meaning of a picture to its graphic representation. By interfacing a semantic network system (SNePS) and an intelligent graphics package (Graflisp), we have gotten a graphics system to reason about abstract concepts relating to displayed objects. Inherent to this system is its ability to create new object hierarchies, realize objects' semantic relationships, compose original scenes, and render into an image form a scene which was requested at a conceptual level. An interesting aspect of this system is that this realization and composition process utilizes knowledge about both general human visual preferences and specific user preferences. This, coupled with the system's ability to represent beliefs of individuals in addition to known facts, makes it a powerful base for an expert graphics system.

8. Work in Progress

In the next phase of this research, we plan to expand the initial experiment for pictures with a larger number of elements and increase the number of functional linkages. This will allow us to include more interesting picture elements and more complex relations among them. For example, we will define and explore three dimensional layout variables for number, distance, rules of occlusion, and angle of viewpoint. These will relate to solid modelling techniques for computer imagery.

At that level, instead of utilizing predetermined representation of the knowledge about the picture, we will have to use machine learning techniques for building and modifying the knowledge base. The automatic rule acquisition program, which we will incorporate into our system, represents knowledge as condition-action rules whose format is based on variable-valued logic calculus, an extension of a many valued logic with typed variables. Programs of a similar type have been tested on a number of practical problems [3]. The results show that, for specific, well defined problems, inductive learning techniques are already powerful enough to discover useful knowledge. Our study seems to be a very good example of this kind of problem.

The rule acquisition program also can be used for discovering display preferences of a particular user or a group of users. This information can help in establishing adaptive criteria that guide the computer graphics system in determining the perceptually optimal graphic representation [4]. The system may even correlate these criteria with perceptual and cognitive factors such as attention and preference. Ultimately our work will result in effective principles for graphic presentation of computer images for specific purposes and certain classes of individuals.

References

[1] Shapiro, Stuart, The SNePS semantic network processing system, Associative Networks, N.V.Findler (ed.), Academic Press, pp179-203, 1979.

[2] Gardner, Brian R., GRAFLISP: A Graphics Package Design for Artificial Intelligence Applications, Masters Thesis, Department of Computer Science, Boston University, 1985.

[3] Michalski, R.S. and Chilauski, R.L., Learning by Being Told and Learning from Examples: An Experimental Comparison of the Two Methods of Knowledge Acquisition on the Context of Developing an Expert System for Soybean Disease Diagnosis, International Journal of Policy Analysis and Information Systems, Vol.4, No.2, 1980.

[4] Holynski, Marek and Lewis, Elaine, Effective Visual Representation of Computer Generated Images, IEEE Proceedings, 5th Symposium on Small Computers in the Arts, IEEE Computer Society Press, 1985.