SEMANTIC NETWORK REASONING FOR PICTURE COMPOSITION

Rafail Ostrovsky
Computer Science Department

Brian R. Gardner
University Professors Program

Marek Hołyński
Computer Science Department *

Boston University, Boston, MA 02215

ABSTRACT

To design an intelligent graphics system, conceptual information has to be represented and reasoned about. This paper explores knowledge representation schema, tools and techniques which are necessary for creating such a system. We will present a system that allows us to relate the meaning of a picture to its graphic representation. Two major blocks of the system are responsible for abstract reasoning about a picture and for picture composition respectively. We will show how a Semantic Network formalism and Semantic Network-based reasoning can be employed to allow abstract reasoning about a picture. How based on conceptual level information, the system reasons about appropriate scene composition and image generation. A smooth transition from conceptual level information to the actual picture composition is also achieved. We introduce a simple environment in which we have tested our approach. With in the constraints of this environment our system reasons about abstract non-visual concepts, decides which physical objects should be displayed, and renders them into an image.

KEYWORDS:
Intelligent graphics system; Knowledge representation; Temporal reasoning; Scene composition.

1. Introduction

Current graphic systems place the tasks of scene composition and object formation on the user. Their performance is limited to the representation of an image that was fully determined by a programmer. The situation would improve when, instead of burdensome testing of different versions of an image, the user could obtain some assistance in deciding about the user could obtain some assistance in deciding about graphic presentation from an intelligent computer graphics system.

In order for a system to provide such assistance, it should have some knowledge about the nature of the objects which user wants to display. In many cases, the user may wish to give only abstract specifications of the picture he wants to see. How does a system decide **what** to display and in what **form** it should be displayed? What kind of knowledge representation and reasoning mechanisms do we need to cope with this problem? How does one go from abstract specification of the picture to the actual image being generated? In this paper we will try to answer some of this questions and present a design of a system which is responsible for the appropriate picture composition and generation.

To create an intelligent graphics system, tools must first be developed which integrate techniques from the fields of Artificial Intelligence and Computer Graphics in a coherent manner [2]. Our system consists of two modules. Reasoning and determination of the conceptual specifications for the picture is facilitated by the Semantic Network Processing System (SNePS) [4]. SNePS allows us to create the conceptual knowledge base in the form of a semantic network and is capable of including rules of inference in the same representation. It decides on the appropriate meaning of the picture and on the objects which should therefore be included in the picture. SNePS passes this information to a graphics module (Graflisp [1]) along with some set of restrictions for the picture composition.

A graphics module is responsible for object generation, transformation and display rendering. Graflisp maintains knowledge of object structures and of inter-relationships between objects. Objects bare a special class inheritance. This allows objects to be comprised of any forms which can be functionally defined, rather than limiting them to be built-up from a polygonal base. Users may define and link their own classes to the classes which are currently defined (polygons, spheres, surfaces of revolution, etc.). In addition to the smooth shading,

* Also, Center for Advanced Visual Studies, Massachusetts Institute of Technology, Cambridge, MA 02139.

realism and perceptual clarity is enhanced by the use of color or texture patterns on the object's surface. The system can reference both the color and texture mapping functions associated to an object at render-time. This allows either the mapping of images (from a camera or a previous image) or of a functionally defined artificial texture onto an object's surface (see **Fig. 1).**

2. Conceptual level

Since we want to deal with various concepts, we have to **represent** them in our system. Therefore, we use a semantic network in which every concept is represented by a node. Such a semantic network formalism has been implemented by S. C. Shapiro [4]. The arcs between concepts represent relations of one concept to another concept. We can talk about the *distance* from one concept to the other, where, the shortest path via some set of arcs as an intuitive "closeness" of two concepts. In general, the meaning of every concept (i.e. node) in the network is defined in terms of the rest of the network.

We will differentiate between abstract concepts and concepts of some physical objects. Whenever a user refers to some abstract concept, a correspondence should be found between this abstract concept and some physical object. To achieve this correspondence the domain-specific knowledge and "distance" factors are used. If the abstract concept corresponds to several physical objects, the system queries the user and stores the answer as a default choice for future reference. Later, if the system is faced with the same choice, the previous decision is made. The user, of course, may override the default.

Sometimes, in addition to the objects directly specified by the user, we want to display some objects which are **semantically** appropriate in the image. For example, if somebody wants to see a drawing of the president's home, he may expect to see related objects in the scene, such as the president in front of the White House. Thus, at a present time, the system should display Reagan in front of the White House. However, if we ask the system to draw the same picture five years later, it will not be Reagan. Two different kinds of inference are present. A **semantic inference** tells us that when we want to see a picture of the White House, it may be appropriate to display the president in front if it. A **temporal inference** tells us that at the present time the president of U.S. is Reagan. We will examine both semantic and temporal reasoning in our system.

3. Puppets World

To study what type of representation and inference is necessary for integrating conceptual information with the actual picture generation we needed to have a domain which does not require a large body of common-sense knowledge. It should be rich enough to have several abstract concepts, several physical objects and in which temporal reasoning can be tested. We have chosen *Puppets World* †. In Puppets World several puppets live in several houses. The system knows a physical description of every puppet as well as a correspondence of each *concept of a puppet* to its 3-D graphical "body". First, we create concepts (i.e. nodes) of several puppets. We tell to the system that every puppet has a place to live and a place to work. Then, we introduce concepts of a *home* and a *workplace.* We assume that during every day of the week each puppet goes to his corresponding work and is at home during any other time. Thus, if the system needs to know who is present at a certain time and a certain location, the temporal reasoning mechanisms will have to be employed.

We give the following information to the system: Ron, Bill and Jane are puppets. There are two locations: the White House and a Regular House. Bill and Jane live at the Regular House. Jane works at home and Bill works at the White House. Ron lives and works at the White House.

We want our system to be able to generate pictures from the following types of queries: "Show me the house of Bill"; "Show me the home of Ron"; "Show me a picture of a puppet and a house"; "Show me a workplace of Jane", etc. We would like the system to decide what various abstract concepts (like home and a workplace) correspond to. We also want our system to display appropriate puppets with every location if they are at this location at the given time. For example, we want the system to display a regular house and Jane if we ask "Show me the home of Bill" during Bill's work-time.

4. Semantic Inference

After the "Puppets World" data is given to the system, we also have to tell the system about the relations among different concepts. The system knows in which house which puppet lives. We can now proceed to specify what the concepts of "workplace" and "home" mean:

---

† This name was given as an analog the the well-known **Blocks World.**

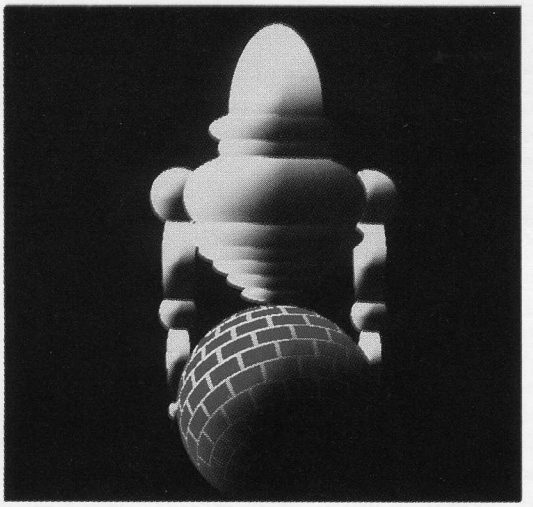


Figure 1. Graflisp example of color mapping, light modelling and hidden surface removal.

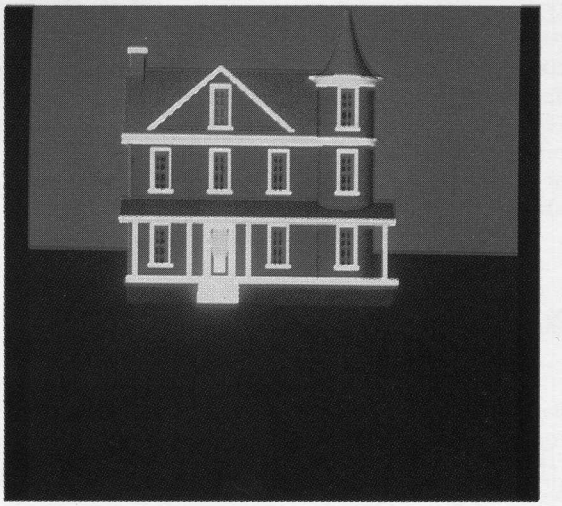


Figure 2. System's response to a request to display a Regular House.

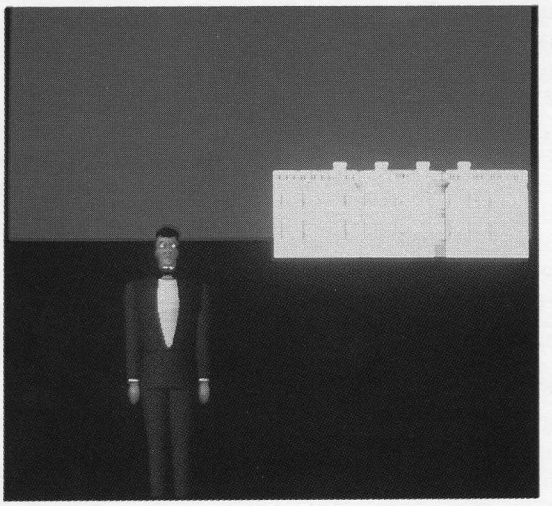


Figure 3. Situation at the White House during off-hours.

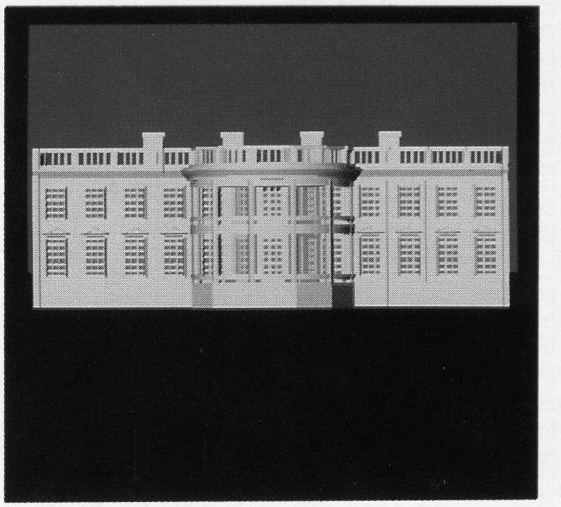


Figure 4. System's response to a user's request to display the White House.

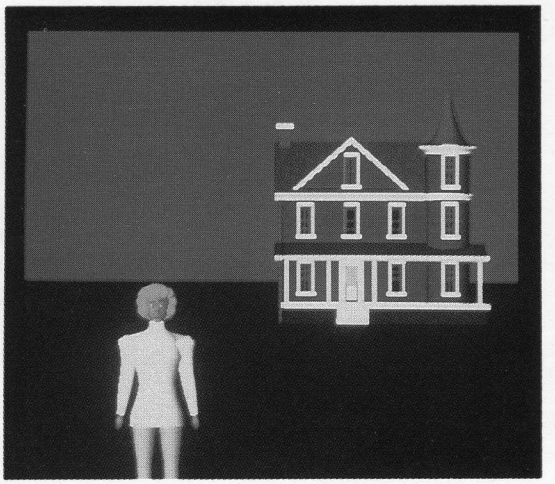


Figure 5. Situation at the home of Bill during business hours.

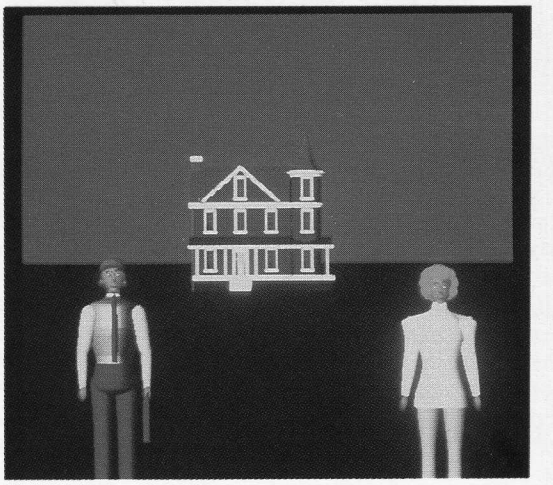
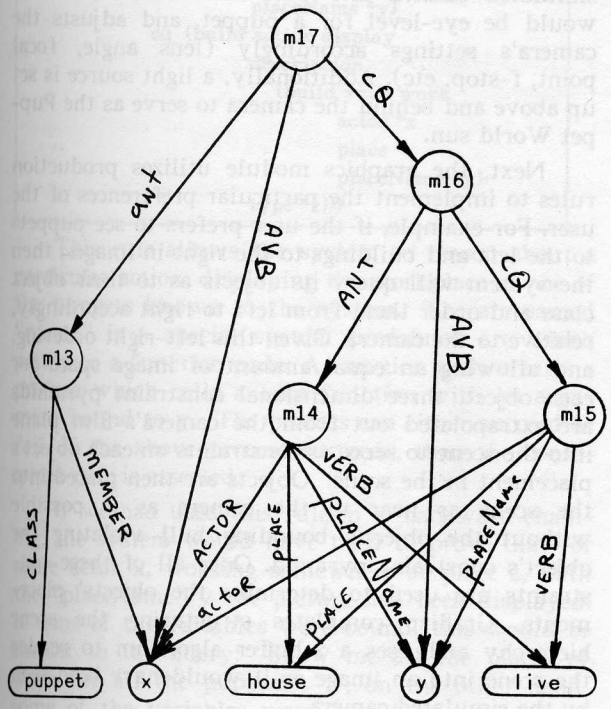


Figure 6. Situation at the workplace of Jane during free time.

```
; For all x
;   if x is a puppet
;     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 puppet)
       cq  (build avb $y
                 ant (build verb live
                           actor *x
                           place house
                           placeName *y)
                 cq  (build verb live
                           actor *x
                           place home
                           placeName *y)))
```



In the above rule "avb" arc stands for "all-variables-bound" and represents a universal quantifier. SNePS uses its rules in both forward and backward reasoning. In this rule, an English description is given first, (lines starting with semi-colons) an actual SNePS User Language code follows and then a semantic network which is actually built. In this semantic network rule, node **m17** represents the entire rule. If the antecedent of the rule has matches in the network (node **m13**), then a consequent of the rule (node **m16**) is executed. But the network under node **m16** is also a rule. Thus, the system looks for matches of proposition represented by the rule node **m14** and asserts the proposition under **m15** with appropriate bindings for **x** and **y**. A similar rule is created to represent the fact that a place where puppet works is his workplace:

```
; For all x
;   if x is a puppet
;     then for all y
;       if y is a house where x works
;         then y is a workplace of x.
```

Since the system has the initial knowledge of places where people *live* and places where people *work*, it will be able to deduce what a reference to somebody's *home* or *workplace* mean. For example, imagine that we ask the system to display a work-place of Jane. The system will first check if Jane is a puppet (which is given), then it will look for a place where Jane works and bind value of **y** to the workplace of Jane. The graphics package will then proceed to display a house, which is where Jane works, realizing that it is the right house to display (see **Fig. 2**).

5. Temporal Reasoning

We can express the rules that all puppets are at work during work-hours and at home during off hours: "If it is a *freeTime*, then a puppet must be at home"; "If it is a *workTime*, then a puppet must be at work".

```
; For all x
;   if x is a puppet
;     then if currentState is workTime
;       then x is at his workplace.

(build avb $x
       ant (build member *x
                 class puppet)
       cq  (build ant (build
                       currentState workTime)
                  cq  (build
                       verb currentlyPresent
                       actor *x
                       place workplace)))
```

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 the particular home or workplace of **x** is. (Two SNePS rules have been created to make this inferences.) To decide whether it is a workTime or a freeTime, 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 or it is a freeTime if and only if a function node "isIt-Worktime" or "isItFreeTime" succeeds. The system uses 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 system solves this problem by removing the temporal assertions each time, thus forcing the system to deduce them again.

We have tested our system by asking it different queries during different times of the day. During "freeTime" we ask the system to display the situation at the home of Ron (**Fig. 3**). During day time we ask the system to show the situation at a home of Bill (see **Fig. 5**). In the evening, we ask the system to display the workplace of Jane (**Fig. 6**).

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 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.

6. Scene Composition

After the system has inferred which objects are actually going to appear in the picture, it is the responsibility of the graphics module to arrange these objects into a coherent scene and produce the image. To do this, first it must gather all of the conceptual constraints to be placed on the image and extrapolate them into three dimensional environmental constraints. The system starts this process by calculating a bounding hull for each object involved; these hulls are used to speed up rough calculations and insure that no two solid objects will occupy the same space.

The graphics module (Graflisp) starts to build an object hierarchy for the scene. The algorithm that it uses is very similar to the way in which a photographer might shoot a picture of a table-top scene. After using the hulls to calculate how much space it will need, Graflisp creates a sufficiently large blue backdrop and a horizontal green surface to serve as the sky and grassy ground for the environment in which it will place the objects. It then places a simulated camera into that environment at what would be eye-level for a puppet, and adjusts the camera's settings accordingly (lens angle, focal point, f-stop, etc). Additionally, a light source is set up above and behind the camera to serve as the Puppet World sun.

Next, the graphics module utilizes production rules to implement the particular preferences of the user. For example, if the user prefers to see puppets to the left and buildings to the right in images, then the system will query its objects as to their object class and order them from left to right accordingly, relative to the camera. Given this left-right ordering, and allowing an equal amount of image space for each object, three dimensional constraint pyramids are extrapolated out from the camera's film plane into the scene to serve as constraints on each object's placement in the scene. Objects are then placed into the scene as near to the camera as is possible without the object's bounding hull violating the object's constraint pyramid. Once all of these constraints are used to determine the objects' placements, Graflisp completes structuring the scene hierarchy and uses a z-buffer algorithm to render the scene into an image as it would have been seen by the simulated camera.

7. Reasoning about the display

How does the inference engine know whether the abstract concepts have a physical counterpart, and if they do, whether the graphics module is capable of displaying them? In order to perform further reasoning about the picture, the system has to have a way to assume that some picture elements have been displayed. A solution is to attach special Lisp functions to our rules, which will query the graphics module about its success or failure. SNePS *function nodes* give a procedural attachment capability to

the otherwise declarative style of SNePS programming. To "prove" a function node, the system must call the 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 working 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 work
          actor *x
          place *z
          placeName *y)
 cq  (build
       ant (build name: showPicture
                placeName *y)
       cq  (build action display
                description
                  (build verb work
                           actor *x
                           place *z
                           placeName *y)
                type *z)))
```

This rule allows the system to reason about a particular action: displaying the workplace of one of the puppets known to the system. The arc named "name:" is a special system predefined arc which points to a function node. A function node creates a process which calls a Lisp function with an argument bound to **y**. The process can either succeed or fail. If the above process succeeds, then the consequent rule is asserted.

If we had used this rule in a backward chaining, the system would have tried to prove that for some actor **x**, working 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 work for all the puppets". If, on the other hand, some of the variables were bound, it would be a reference to some specific instance of **x** , **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 works". To prove this, we would have to prove that the entire nested rule is valid. This rule would create a SNePS process, which would call Graflisp. When Graflisp displays the picture, the function node succeeds and returns true, and then the final consequent would be asserted. An English description of the rule for displaying a puppet's living place is provided below.

```
; 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.
```

For example, if we ask the system to show the house where Ron lives, the White House will be displayed (see **Fig. 4**).

8. Conclusions

The developed system allows us to relate the meaning of a picture to its graphic representation. Its two major blocks facilitate the intelligent computer graphics system's needs for reasoning about the content of a picture, the picture composition and the image generation. By interlinking SNePS and Graflisp, we have been able to obtain images originating from abstract requests. We have introduced a simple environment consisting of several puppets living and working in several places (called Puppet's World). Using abstract concepts (such as a home or a workplace) our system has successfully deduced which objects *should* be displayed and has displayed them.

In the next phase of this research, we plan to incorporate default display rules of user preferences. To collect and generate such rules, we will use the system described in [3]. We plan to develop an interactive visual test generator and rule acquisition package which can be used to customize default display rules to the preferences of a particular user as well as to the preferences of different classes of users.

References

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

[2] Hołyński Marek, Brian R. Gardner and Rafail Ostrovsky, "Towards Intelligent Computer Graphics System", *Technical Report*, Boston University, 1986.

[3] Hołyński 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.

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