# Extending Cognitive Architectures with Spatial and Visual Imagery Mechanisms

by

Scott D. Lathrop

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2008

Doctoral Committee:

Professor John E. Laird, Chair
Professor George W. Furnas
Professor Richard L. Lewis
Associate Professor Thad A. Polk

For Susan, Sarah, and Stephanie

# Acknowledgements

I would like to thank Sue, my wife and best friend, who encouraged me to pursue a PhD and then backed those words of encouragement with tremendous love and support. Thank you for supporting me by doing more work than a mom should ever have to do. I would not have accomplished this goal without you! Thank you, Sarah, my five-year-old daughter, who while setting the table for dinner one night, gave me the idea of how a child may use imagery to perform such a task. Sarah always kept my research in perspective with her many questions such as whether the "story" I was writing had anything to do with the "game" I made, and whether or not I was going to talk about it at "show-and-tell". To Stephanie, my two-year old daughter, who despite instructions not to enter the study when the door was closed, entered anyway, ran to me, and give me a big hug. Those moments are precious and always brightened my day even when I had not accomplished much. Sarah and Stephanie's games of "hide-and-go-seek" also inspired some of my thoughts. To my parents, thanks for always encouraging me to "climb the next mountain", work hard, and persevere.

A special thanks to my advisor, John Laird, who helped cultivate these research ideas and then provided me with the latitude to develop them. As a "nontraditional" graduate student, John respected my professional background and kept me on track by encouraging me to focus on the most important ideas, start simple, and always evaluate. To my committee members, George Furnas, Rick Lewis, and Thad Polk, thank you for encouraging me to think about depictive representations (George) and to consider the psychological and neurological constraints (Rick and Thad). I enjoyed the discussions and learned a lot at the biologically inspired cognitive architecture (BICA) meetings. I appreciate my fellow Soar graduate students--Andy, Bob, Jakub, Joseph, Nate, Nick, and YJ--for the friendship, stimulating discussions, and letting an "old guy" hang around the research lab.

# Table of Contents

# List of Figures

# List of Appendices

# Chapter 1

# Introduction

The generality and compositional power of sentential, symbolic structures has made it central to knowledge representation and processing in cognitive architectures. General, cognitive architectures have proven useful in modeling mental processes for both scientific, psychological research and real-world applications such as simulation entities and robots. However, cognitive architectures have failed to address and account for inherently perceptual and modality-specific thought processes that some argue should participate directly in thinking rather than serve exclusively as a source of sensory information (Barsalou, 1999; Chandrasekaran, 2006). Spatial and visual imagery are examples of such thought processes.

With a few exceptions (Kurup, 2008), there has not been a proposed coherent system from the Artificial Intelligence or Cognitive Science community that integrates and uses symbolic and perceptual representations for reasoning. Current cognitive architectures use metric representations, but for control, and not for representing and manipulating task knowledge (Anderson et al., 2004; Laird, 2008; Sun, Slusarz, & Terry, 2005). No architecture reasons with visual depictive representations. Metric and depictive structures may serve as perceptual input but not as first-class knowledge structures that an agent can use for inferring new knowledge.

In this research, we explore the utility of general-purpose, intelligent systems supporting mechanisms to encode, compose, manipulate, and retrieve symbolic and

perceptual-based representations. In addition to the traditional *symbolic* representation, the resulting architecture uses *quantitative spatial* and *visual depictive* representations that serve as a basis for spatial and visual imagery processing. Behavioral and biological constraints, primarily derived from Kosslyn (1980; Kosslyn, 1994; Kosslyn, Thompson, & Ganis, 2006), and computational constraints influenced by Newell (1990), inform the architectural design. From a theoretical standpoint, we account for high-level vision, as spatial and visual imagery are dependent on both cognition and visual perception (Finke, 1989; Kosslyn, 1994; Kosslyn, Thompson, & Ganis, 2006; Palmer, 1999; Peronnet, Farah, & Gonon, 1988; Podgorny & Shepard, 1978). At this point, underlying visual processing algorithms are ad hoc and do not model the details of human performance, but modeling the interdependence facilitates an enhanced understanding of the constraints imposed between the thought processes.

The work reflects a computational synthesis of spatial and visual imagery, visual perception, and cognition within the computational constraints of the Soar cognitive architecture (Laird, 2008). Empirical results from three different domains illustrate the computational gain and functional value the resulting architecture achieves. The results show how specialized, architectural components processing quantitative spatial and visual depictive representations can achieve an order of magnitude (or more) speed up over traditional symbolic processing without trading off generality. Furthermore, the architecture demonstrates new functional capability and improved problem-solving quality when using imagery in tasks rich with spatial and visual properties.

The psychological basis for the research is mental imagery processing. Humans use mental imagery to assist them with reasoning, problem solving, decision-making, creativity, learning, and motor rehearsal (Helstrup, 1988). Some attribute creativity to the observation that one can combine objects in their mental images to reveal novel objects and relationships (Finke, 1989). Athletes often report using imagery or "visualization techniques" to rehearse their motor skills prior to competition. People with certain forms of autism report relying almost exclusively on imagery in their thought processes (Grandin, 2006). Even to learn non-visual concepts, such as "love," they must have a visual, concrete representation of the concept (e.g. a heart). This research focuses on

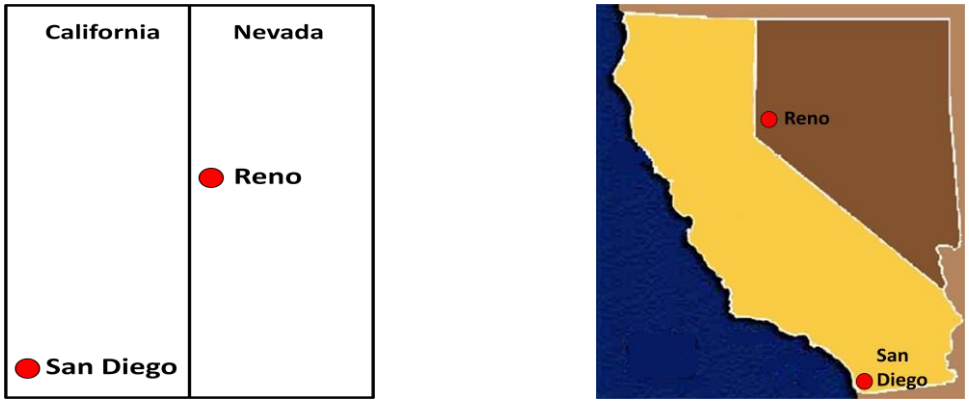
using two forms of mental imagery, spatial and visual, for reasoning, decision-making, and problem solving.

*Spatial imagery* assists humans with spatial reasoning tasks. For example, imagine that you are facing south and we ask you to simulate the following movements: step forward, turn right, step forward, turn right, and step forward. If we then ask you what your final location and orientation is, most people can respond that they are to the left of where they started facing north. This task requires you to infer a global spatial relationship (i.e. your final location and orientation) from a set of local spatial relationships provided in the task instructions. Your shape or contour characteristics as well as the shape of the area you are stepping are not relevant to the problem.

Humans rely on *visual imagery* for the detection of spatial and visual properties not previously encoded as symbols where the specific shape or color of object(s) is necessary for the inference. For example, consider how you answer the following questions. Does the letter 'A' have an enclosed space? What is the shape of a dog's ears? What is wider in the center, Michigan's lower peninsula or the state of Ohio? When asked, most people respond that they create a visual depiction of the object(s) and then "look at" the image to answer the question. This type of reasoning requires a depiction because the inference is directed at a feature (e.g. enclosed space) or spatial property (e.g. width) requiring specific shape to formulate an answer. On the other hand, if asked, "What state is larger geographically, Alaska or Rhode Island," most American adults can formulate an answer without having to create an image. They know from previous study that Alaska is the largest state and Rhode Island is the smallest state, which is a general fact easily encoded with symbolic representations.

General problem solving may use all three types of representations. If you are at a furniture store trying to choose a new sofa for your living room, you may imagine your living room to see if the shape and color of the furniture matches your current decorum (visual imagery) and possibly simulate moving your furniture around to "see" where the new sofa fits best (spatial and visual imagery). On the other hand, sometimes you use spatial imagery to form a general answer and then, if given time (or prodding), use visual imagery to form a more accurate inference. Consider another geography question. What

city is further to the west, San Diego, California or Reno, Nevada?[1] Initially, you might build a quantitative spatial structure representing California and Nevada as geometric shapes (such as rectangles). Then using symbolic, factual knowledge that California is to the west of Nevada, arrange the geometric figures accordingly (Figure 1-1a). Again based on your knowledge that San Diego is in the southwest corner of California and Reno is in the west central section of Nevada, you place "dots" in those locations. You then reason that San Diego is west of Reno. This answer is a common mistake people make. However, if you add more "detail" by adding the specific shape of the two states (visual depictive), you can correctly "see" that Reno is west of San Diego (Figure 1-1b). Note that even though we illustrate the quantitative spatial representation as a picture in Figure 1-1a, we can represent the objects (California, Nevada, San Diego, and Reno) as a set of points in a Cartesian coordinate system—a picture is not required. However, when we add shape to the specification (i.e. the shape of the states), an image is a more suitable structure where space is inherent in the representation.




**(a) Quantitative Spatial Representation** **(b) Visual Depictive Representation**

**Figure 1-1: Representations Involved in Spatial and Visual Imagery**

What the previous examples illustrate is the power of being able to reason by combining these representations through imagery processing. Of course, to perform these tasks assumes:

[1] Example from (Stevens & Coupe, 1978)

- You have previously encoded a representation of each object's shape and color and stored it somewhere in memory (e.g. letter, states, or furniture).
- You have previously encoded local spatial relationships between pairs of objects (e.g., California is west of Nevada, Reno is in Nevada, your sofa is in front of the T.V.). Note that the types of spatial relationships include direction (west, in front of), distance (number of steps), orientation (the orientation of the objects), topology (in), and size (sofa is larger than chair).
- You recognize your current state and goal (e.g. determine if the letter "A" has an enclosed space, determine what direction you are facing, decide what furniture you should purchase, etc.).
- You are able to combine these different forms of knowledge and retrieve the desired information.
- You can access the results of your retrieval to make a decision and form a response.

These issues are the types we explore in this research to determine the computational mechanisms underlying spatial and visual imagery. The goals and relevant questions of our research follow:

(1) To incorporate spatial and visual imagery within the context of a cognitive architecture inspired and constrained by behavioral, biological, functional, and computational evidence.

- What are the representations and processes that are architectural?
- What knowledge is necessary to create these representations and control the processing?
- What is the relationship between spatial imagery, visual imagery, and visual perception? What underlying structures and mechanisms do they share? What components are unique to imagery?
- Where is information stored and in what representational format?
- Where is the information processed?
- What information is transmitted between the architecture's functional components?

(2) To understand spatial and visual imagery's capabilities and limitations.

- How does cognition use spatial and visual imagery to solve problems? What are the types of problems?
- What are the environment and task conditions where spatial and visual imagery processing provides additional functional capabilities?
- What types of tasks are computationally more efficient using a quantitative spatial or visual depictive representation versus using a symbolic representation? What are the tradeoffs?

(3) To expand the integration between perception and cognition. We are focused specifically on how cognition uses perceptually based representations for reasoning and problem solving rather than how the system processes bottom-up perception into symbols (e.g. computer vision, robotics) or how perception constrains the timing, processing, and control behavior of the architecture (e.g. EPIC, ACT-R).

(4) To determine and build appropriate tools for debugging and evaluating a spatial and visual imagery component within a cognitive architecture (software engineering aspect).

As further clarification, the following is a list of what are ***not*** our research goals.

(1) Detailed modeling of human behavior (Cognitive modeling). We are using psychological theories, experimental evidence, and neuropsychological results as inspiration in our architectural design. We would like the system to exhibit the general behavior and be plausible in accordance with how we believe humans solve problems using spatial and visual imagery. At this time, however, we are not concerned with matching human experimental results.

(2) Building a stand-alone model of mental imagery. We are not trying to model mental imagery without taking into consideration how it fits into the overall architecture. Our goals are much more general in that we want to discover how the different representations are used for problem solving. The system has to work together as a whole with spatial and visual imagery processing as one of cognition's possible tools.

(3) Designing and evaluating specific algorithms or attempting to claim we have all imagery functionality implemented. As a follow-up from the previous point, the scope of this work is general so that attempting to analyze, design, and evaluate the details of specific algorithms would take away from our focus.

We organize the remainder of this dissertation as follows. Chapter 2 and 3 present our design space constraints and theory. We devote Chapter 2 to the discussion of the quantitative spatial and visual depictive representations, as they are central to our theory.

The chapter includes a summary of the *mental imagery debate* and its influence on our decisions. Chapter 3 presents the remaining design space constraints and summarizes our theory. Appendix A provides additional background on the relevant psychological and neuroimaging experiments supporting our theory.

Chapter 4 compares previous work in Artificial Intelligence and Cognitive Science. Included in this chapter are computational approaches that have either modeled mental imagery or used it as motivation for a specific application. Chapter 5 summarizes task and environmental characteristics where spatial and visual imagery is useful. The chapter also presents an overview of our three experimental domains to facilitate the architectural discussion in Chapter 6. The three experimental domains include an agent using spatial imagery to solve a geometry problem, visual imagery to recognize features on individual alphabet letters, and both spatial and visual imagery to inform its decision making in a simulation of an Army small-unit leader. Chapter 6 discusses the memories and processes associated with the architecture that include Soar and its Spatial-Visual Imagery (SVI) component. Appendix B and Appendix C provided more details on the design and implementation of the system. Chapter 7 provides the subjective and objective evaluation of the architecture across the three experimental domains. The evaluation metrics include behavioral, biological, functional, and computational design space constraints; computational gain; functional capability; and problem-solving quality. We conclude with our research contributions and future work in Chapter 8.

# Chapter 2

# Spatial and Visual Imagery Representations

A key result from mental imagery experiments is that humans use multiple types of representations during imagery processing. As the distinction between these representations is central to our theory, we focus exclusively on them in this chapter. We begin by summarizing the three representations that support spatial and visual imagery processing and discuss their functional and computational tradeoffs. We then summarize the mental imagery debate. The debate is important to understand as it directly influences the decision as to whether a cognitive architecture should include separate mechanisms for spatial and visual imagery. The alternative is to assume that symbolic cognitive architectures are sufficient for imagery processing and what an agent requires is simply additional knowledge.

## 2.1 Symbolic, Quantitative, and Depictive Structures

From a functional and computational perspective, our hypothesis is that spatial and visual imagery use at least three distinct representations to include (1) a *symbolic*, (2) a *quantitative spatial*, and (3) a *visual depictive* representation (Figure 2-1). The symbolic representation (first row of Figure 2-1) is the amodal, stable medium useful for general reasoning (Newell, 1990). Symbols may denote an object, visual properties of that object, and spatial relationships between objects. They are sentential, or sentence-like, in that their meaning is dependent on context and interpretation rather than their spatial

arrangement. The power of symbols comes from their composability using universal and existential quantification, conjunction, disjunction, negation, and other predicate symbols. For example, the right-hand column of Figure 2-1 represents two objects, a can and a box with symbols denoting visual features (e.g. (can, yellow)) and spatial properties (e.g. on(can, box)). In addition to visual and spatial properties, symbols can represent non-visual or non-spatial content, which is necessary for associating an object with other modalities and concepts.

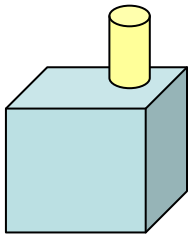
| Representation | Aliases | Modality | Processing | Uses | Example |
|---|---|---|---|---|---|
| **Symbolic**<br>• Visual Properties (optional)<br>• Spatial Properties (optional) | Sentential<br><br>Propositional<br><br>Descriptions<br><br>P-Symbols | Amodal | Symbolic manipulation<br><br>Logic<br><br>Entailment | *General Reasoning*<br><br>Explicit visual feature recognition<br><br>Qualitative Spatial Reasoning | object (can)<br>feature (can,curve)<br>color (can, yellow)<br><br>object (box)<br>feature(box,corner)<br>color (box, blue)<br><br>on (can, box) |
| **Quantitative spatial**<br>• Visual Properties<br>○ General Shape (inferred from size dimension)<br>• Spatial Properties (mandatory)<br>○ Direction / Distance (location)<br>○ Orientation<br>○ Size<br>○ Topology | Sentential<br><br>Metric<br><br>Diagrams<br><br>P- and I-Symbols<br><br>Perceptual Symbols | Amodal | Mathematical manipulation<br><br>Laws of Dynamics (motion) | *Spatial Imagery*<br><br>Spatial Reasoning *(General shapes)* | can<br>location <2,1,2><br>orientation 0<br>height 5, radius 1<br><br>box<br>location <0,0,0><br>orientation -10<br>length 10<br>width 6 height 4 |
| **Visual depictive**<br>• Visual Properties<br>○ Shape, Color<br>○ Explicit features<br>○ Explicit empty space<br>• Spatial Properties<br>○ Direction / Distance (location)<br>○ Orientation<br>○ Size<br>○ Topology | Iconic<br><br>Analog<br><br>I-Symbols<br><br>Perceptual Symbols | Visual | Mathematical manipulation<br><br>Depictive manipulation | *Visual Imagery*<br><br>Visual Feature Recognition<br><br>Spatial Reasoning *(Specific shapes)* |  |

**Figure 2-1: Imagery Representations**

The quantitative spatial representation (second row of Figure 2-1) is also amodal, but is perceptual-based. That is, it is an interpretation of visual, auditory, proprioception, and kinesthesis senses based on a fixed frame of reference that asserts an object's location and orientation in space. The frame of reference can be relative to an agent's viewpoint (egocentric) or another object (allocentric). Computationally, the structure uses scalar values and vectors in a three-dimensional Euclidean space to represent information with symbols labeling the objects. The processes that infer information from this structure use sentential, mathematical equations.

Spatial imagery uses the representation for spatial reasoning and simulating motion through linear transformations (i.e. translating, rotating, and scaling) or laws of dynamics. The second example in Figure 2-1 represents the metric location, orientation, and size dimensions of the can and the box. Location is a combination of direction and distance from a fixed frame of reference. One may infer rough estimates of size and topology based on the general convex shape, or dimensions, of the objects.

In contrast to symbols or quantitative spatial representations, both of which are sentential structures, space, including empty space, is inherent in the visual depictive representation (third row of Figure 2-1). The depiction is from a privileged viewpoint, and the spatial structure of the patterns resembles the objects in a perceived or imagined scene (Finke, 1989). Computationally, it is an image data structure where the processing uses either mathematical manipulations (e.g. filters, rotation, and scaling) or algorithms that take advantage of the topological structure and color of the representation (Funt, 1976; Furnas, 1990; Furnas et al., 2000).

Visual imagery uses the depictive representation for extracting an object's visual features (e.g. lines, curves, enclosed spaces, corners) or for spatial reasoning where non-convex shapes are inherent to the problem. It also facilitates the identification and location of empty space between objects, exits between scenes, and topology between objects. Similar to spatial imagery, visual imagery can use the depictive representation to simulate physical processes. For example, visual imagery processing can simulate moving the can in Figure 2-1 to the right to determine where it falls off the box (Funt, 1976).

Even though we only label the top structure in Figure 2-1 as a symbolic representation, each of the representations are symbols in the sense that each is a pattern denoting something where "denotation is a mapping of patterns onto their meanings" (Simon, 1996). Although we typically think of symbols as linguistic patterns and reasoning as logic or mathematical equations, non-linguistic patterns, such as depictions, are also symbols, but the reasoning processes that infer information from it are not based on logic. In symbolic terms, the distinction between the representations is that some are pointer symbols (P-Symbols), such as what we call symbolic representations, and some are information symbols (I-Symbols), retinal input, for example, being an extreme case. Pointer symbols do not contain raw information, but rather serve as an abstraction of more detailed information. Information or perceptual symbols (Barsalou, 1999), are carriers of information where the encoding pattern is primarily raw information such as in the depictive representation. Hybrids, such as the quantitative spatial representation, contain both P- and I-symbols.

## 2.2 Functional and Computational Tradeoffs

So why does a cognitive system use these three representations during thought processing? In short, each structure has functional and computational tradeoffs. From a functional perspective, there are tradeoffs between the representations that a specific task often highlights even when the environment remains constant. For example, given appropriate inference rules and the symbolic representation in Figure 2-1, one can infer that there is a yellow object (can) on a blue object (box). However, one cannot infer that the top of the can is a circle. One can infer visual properties from a symbolic representation only when the property is encoded explicitly as a symbol or when task knowledge supports the inference (e.g. if two lines intersect then there is a vertex).

Consider another example from Ullman (1996). Figure 2-2 shows two enclosed regions. Assume that the region in Figure 2-2a is a quantitative spatial representation rather than the image shown for presentation purposes. That is, assume the region is a set of x, y points with indices specifying the connections, or line-segments, between the points. One way to determine if the dots in the figure lie inside or outside the enclosed region is to imagine a ray (again as a quantitative representation) from the point to

"infinity" and then count the number of intersections between the ray and the line-segments making up the region. If the count is odd, then the point is inside the region; otherwise, it is outside the region.

**(a)** **(b)**

**Figure 2-2: Example of the Capability and Limitation of Representations**

Now consider the situation in Figure 2-2b where the environment is the same, but the task is to determine whether the two points lie in the *same* region. Using the same ray-based methodology will not provide the desired information as the number of intersections only tell you whether the point is in a region or not. It does not tell you *what* region (if there is more than one region). Rather than using rays to determine the region, assume now that the figure is a visual depictive representation (as presented). Start at one of the points and imagine "coloring" all the white area red until you reach a black boundary. If after coloring, red pixels surround the other point, then the two points are in the same region. Such a coloring or activation scheme is similar to the depictive algorithms we use in our architecture. In this example, the environment remains the same, but the task changes requiring a different representation to achieve the desired functional capability.

From a computational perspective, the tradeoff is between scope (what it can represent) and processing cost (Newell, 1990; Simon, 1996) or alternatively, what Norman (2000) categorizes as discretion and assimilability. Symbolic representations are high in terms of discretion as they convey only the intended information required for general reasoning, nothing more or less. They purposefully leave certain aspects of the description indeterminate. The predicate description, "on(can, box)," is sufficient for general inferences used in logical reasoning. That is, you can assert very general statements such as "if the can is on the box then grasp it." In terms of capacity alone, symbolic representations transmit much less information than visual depictions. The

symbolic representation in the right-hand column of Figure 2-1 is roughly $2^9$ (512) bits while the picture of the can on the box is $2^{19}$ (512K) bits—three orders of magnitude more. Symbols may then provide a much more compact structure allowing us to retain context while reasoning and bring in details as required.

Symbols can also represent uncertainty such as "the can is on the box *or* on the floor," and negation as in the statement "if the can is *not* in the hand but is on the box *or* on the floor then grasp it." The quantitative spatial and visual depictive representations have to commit to a particular configuration and so cannot convey these general statements. Over specification is a disadvantage for perceptual representations as any learning may apply only to the particular situation. It is difficult to generalize and transfer to other tasks. A simple example illustrates this point. Assume an agent learns by imagining an 'A' that does not have any curves. Now it can assert that the particular 'A' it imagines does not have a curve, but it cannot assert that *all* 'A's' do not have curves (e.g. consider a cursive letter, $\mathscr{A}$).

At the other extreme, depictive representations are low in terms of discretion (i.e. they provide many details), but for visual and spatial properties are computationally easier to assimilate. For example, from the picture in Figure 2-1, information such as the top of the can looks like a circle and covers about an eighth of the box is directly accessible. What we lose in representational scope, or expressiveness, we often gain in fewer processing cycles as we can exploit the space and color in the image to infer the visual and spatial properties.

In terms of discretion and assimilation, the quantitative spatial representation falls in between the symbolic and depictive representations. It provides more details than the symbolic representation (i.e. direction, distance, orientation, size, and general topology) but less information (i.e. specific shape and color) than the depictive structure. A strip map of the New York City subway system is a good example. It leaves out the details of every turn and provides you with the general topological structure, direction, and distance information. In the middle, quantitative spatial example from Figure 2-1, you can infer the direction and distance between the center of the can and the center of the box, the location where the bottom of the can touches the box, and the relative size between the

two objects. The depiction also provides this information but at the cost of requiring greater capacity.

There are two other computational reasons why the quantitative spatial representation is useful. First, there are some general spatial reasoning tasks where reverting from symbolic to metric information is necessary to infer new information (Edwards & Moulin, 1998; Forbus, Tomai, & Usher, 2003; Mukerjee, 1998). Forbus, Neilsen, and Faltings (1991) coin this lack of a general, purely qualitative representation of spatial properties as the *poverty conjecture*. Second, Marr (1982) stresses that bottom-up visual processing uses incremental, increasingly abstract levels of representations. This rationale is also pertinent to imagery but in the "opposite" direction. Visual imagery cannot generate a depictive representation directly from qualitative symbols without first specifying metric properties, such as the location, orientation, and size of objects as the generation process requires this information to project the shapes to a depiction.

The power of imagery processing emerges from the ability to combine the symbolic, quantitative, and depictive representations, taking advantage of the representation that provides a computational advantage or a specific functional capability. The support for this ability in a general, cognitive architecture is a major contribution of this research. One of the difficulties, however, is deciding when to use the appropriate representation. Although we provide hints throughout this thesis of where each representation is useful, we do not offer a conclusive theory. Therefore, our theory states that an agent, through procedural knowledge, decides which representation to use based on its current state and its estimate of the total cost of using the representation (to include transforming to that representation, using it, and then transforming back) is less than an alternative.

One of the main criticisms of theories advocating the use of multiple representations is exactly this issue. As Simon (1996) articulates:

> "Transformation from one symbolic representation to another, in order to find one that is computationally efficient in dealing with a particular class of problems, is an essential, and little understood process in much problem solving."

The mental imagery debate highlights this criticism, so we will look at the issue from this perspective and articulate our reasons for incorporating both spatial and visual representations into the resulting architecture.

## 2.3 Mental Imagery Debate

Although to the casual observer the mental imagery debate may seem ludicrous (of course we have "mental images"), it is actually quite complicated once one investigates the details. Few deny that when we engage in imagery we *seem* to be forming pictures in our heads. The question is, are we really?

Philosophy and psychology have a long history of mental imagery theories (Tye, 1991). Past theories have cast the role of mental imagery in thought processing at both ends of a spectrum. At one end, philosophers such as Aristotle, Descartes, and Hobbes, advocated visualization as the focal point of thought. They believed mental images were models of the external world. Introspection, or the process of explaining your internal thought processes, was the dominant method in the formulation of these ideas. In the early twentieth century the behaviorist movement, led by Watson (1913), rejected introspection as a valid methodology, arguing that imagery is simply a dramatization of what is actually occurring in a person's mind. He concluded that introspection is not evidence of mental structures and processes.

As cognitive psychology emerged in the 1960s, some began to argue that there must be mental representations used in imagery to explain the results of so many experiments. Hebb (1968) asserted that descriptions of someone's imagery experience is not necessarily introspection. As an example, he used the case of amputees, who after removal of an extremity, report pain and sensation in their "phantom limb." This reporting of sensation is not introspection, he argued, but an imagined experience where the perception originates in a higher brain process rather than from the extremity. Hebb described imagery as the activation of cell-assemblies previously formed during perception. According to Hebb, vivid imagery is the activation of the lower order cell assemblies while higher order cell assemblies are the basis for "less specific" imagery. The separation between vivid and "less specific" imagery is analogous to our theory of visual depictive and quantitative spatial representations.

This change in attitude concerning the relevancy of reported imagery experiences coupled with behavioral experiments that were more scientific, solidified imagery's role in cognition. The major question that remains, and what psychologists and neuroscientists have debated for over three decades is the representation of these internal images. The debate focuses primarily on visual imagery although the discussion of spatial imagery has recently emerged as the theorists refine their theories. Kosslyn is the protagonist for the *depictive* theorists who embrace the notion that visual images are quasi-pictorial, have an inherent underlying spatio-analogical representation, and share similar mechanisms with vision (Kosslyn, 1980; Kosslyn, Thompson, & Ganis, 2006). At the other end of the representational spectrum, there are those, such as Pylyshyn (1973; Pylyshyn, 2002), who argue that there has not been enough evidence to reject what he calls the "null hypothesis." That is, visual imagery uses the same propositional (i.e. symbolic) representations and processes as general, higher-level reasoning. The only difference, he contends, is that the content includes visual and spatial information such as shape, color, direction, and distance. Throughout the debate, cognitive scientists such as Anderson (1978) also raise the key point that a representation is dependent on the computational processing. Any theory must articulate how the representation facilitates the processing and what the tradeoffs are in terms of functional capability and computational efficiency.

The modern debate began after Shepard and Metzler (1971) published their seminal work on mental rotations. Their experiments showed subjects pairs of three-dimensional, non-standard objects and asked them to determine if the objects were the same shape (Figure 2-3). Some pairs were identical but with one of the objects rotated at a different angle than another. Other pairs were mirrored reflections of one another so could not be matched by rotating. After being shown a pair of objects, subjects responded as to whether they thought the objects were the same. Shepard and Metzler found that response times were linear with the rotation angle. Furthermore, the subject's post-experiment reports claimed that in order to make the comparison they had to "mentally rotate" one of the objects. These two pieces of evidence lead Shepard and Metzler to hypothesize that there is some sort of an imagined mental rotation process in three-dimensional space. Some psychologists began describing the phenomena using a

"picture" metaphor to describe the representation and a "mind's eye" as the process that "looks at" the "picture" to infer information.

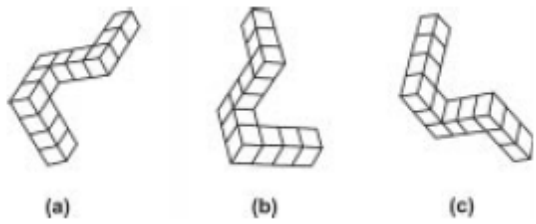



**Figure 2-3: Examples Shepard and Metzler Used to Show "Mental Rotation"**

Leveraging Newell and Simon's (1972) ideas that human problem solving uses symbolic computations, Pylyshyn, wrote a strong argument against the "picture" metaphor for mental imagery. He suggested that imagery, like other thought processes, is amodal and symbolic. In the same vein as Watson (1913), he questioned experiments relying on introspection stating that the images in our head are epiphenomenal. He also questioned the notion of a "mind's eye" arguing that it is really a question of infinite regress. That is, if there is a "mind's eye" then does that imply there is a "brain" for the "mind's eye?"

Kosslyn and Pomerantz (1977) countered Pylyshyn's arguments using empirical evidence and theoretical comparisons between their depictive account of visual imagery and a propositional account. For example, in one of the experiments subjects were presented with a map of a fictional island and seven objects (lake, well, beach, etc.) located at various places on a map (Figure 2-4). They asked subjects to study the map, close their eyes, mentally picture it, and compare their visual image with the map. Once the subject had the map adequately memorized, they were instructed to close their eyes and imagine one of the locations (e.g. "well"). Kosslyn and Pomerantz then named another object (e.g. "tree"), and the subjects were instructed to "scan" to the named object. Kosslyn and Pomerantz measured response times and found that the time to scan between pairs of objects was linear with respect to the distance between objects. They concluded that a visual image preserves distance and space. Symbolic accounts, Kosslyn and Pomerantz argued, cannot adequately explain these findings.

Later, Kosslyn shifted the basis of his argument for depictive representations from behavioral experiments to neuroimaging evidence (Kosslyn, 1994; Kosslyn, Thompson,

& Ganis, 2006). In monkeys it is known that the primary visual cortex[2] (Figure 2-5) roughly preserves the spatial structure of the image on the retina (Tootell et al., 1982). That is, space on the cortex represents space in the world. Kosslyn and others asserted that if the visual cortex shows similar activation patterns during visual perception and visual imagery, then there is a strong indication that visual imagery, similar to vision, is using the topographically mapped or depictive, areas of the brain.

**Figure 2-4: Fictional Island Map**

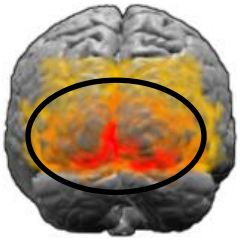


**Figure 2-5: Visual Cortex**

The typical methodology used during these neuroimaging experiments consisted of two groups of subjects. One group would perform a task using vision, and the second group would perform the same task using imagery. During evaluation, response times and brain activity was measured using positron emission tomography (PET) or functional magnetic resonance imaging (fMRI).[3] For example, Kosslyn et al. (1993) had subjects either view (vision group) or imagine (imagery group) a letter in a grid (Figure 2-6). An 'x' then appeared on the grid and subjects had to indicate (by pushing a button) whether

[2]Also known as the striate cortex, V1, or Brodmann area 17. It is the first area of the brain to receive information from the retina.

[3]Positron Emission Tomography (PET) and functional magnetic resonance imaging (fMRI) are medical imaging techniques used to measure neural activity in the brain.

the ‘x’ fell on or off the letter. They also had a baseline group (sensory/motor group) who simply pushed a button when the ‘x’ disappeared from the grid to rule out any activation effects caused by sensing and responding. The results showed greater activation of visual cortex during the visual imagery task than during the visual perception task. They also found more activation in other brain areas[4] indicating that there were additional mechanisms involved in generating the image.

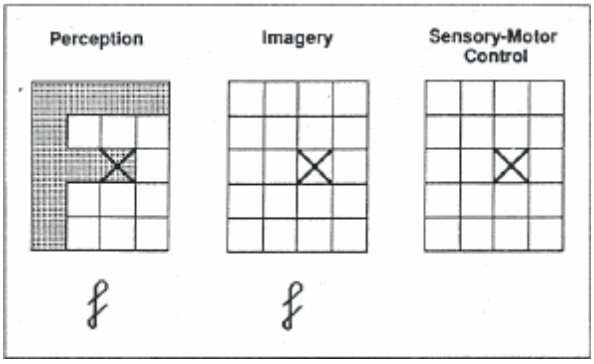



**Figure 2-6: X On/Off Letter Experiment.**
**Subjects either saw a letter in a grid (visual perception task), visualized the letter in the grid (imagery task), or waited for the ‘X’ mark to be removed (sensory/motor task).**

During this time Pylyshyn (Pylyshyn, 1981, 2002) was also active in the debate asserting that *tacit knowledge* and not *architectural* constraints (in the sense of another representation) explained depictive theorists behavioral experiments. For example, he argued that the reason response times for scanning between various imagined objects on the island map were linear with respect to distance was not because the imagery medium uses space to represent, but rather because the participants were instructed to “scan” between pairs of objects. To test his theory, Pylyshyn ran a similar “island map” experiment. First, he instructed subjects to memorize a map and refer to their mental image of the map. Next, they were instructed to close their eyes and imagine one of the locations. Pylyshyn then named another object on the map and instructed the subjects to determine the compass direction (NE, N, NW, W, SW, S, SW, and E) from the second object to the first object. The task instructions did not provide an indication as to how the subjects were to determine the direction (i.e. there was no instructions to “scan”).

[4]Specifically Broadmann area 44, 45, and 46. Area 46 is functionally part of a group known as the dorsolateral prefrontal cortex (DLPFC) and hypothesized to be responsible for executive functions.

Pylyshyn's results showed no correlation between response time and distance between the objects.

Pylyshyn argued that if the task demands (i.e. "scan") alter the behavioral pattern (in this case the response time), then knowledge explains the results, not the underlying architecture. He calls this phenomena *cognitive penetrability*. When we are told to "scan" our image, he argued, it takes a certain amount of time until we arrive at the next object because knowledge of how long it takes to scan a specified distance controls, or mentally simulates, our scan rate. The intrinsic properties of the architecture are not involved, only knowledge.

Furthermore, Pylyshyn claimed that there was not enough evidence to show conclusively that a human's visual cortex is a topographically mapped representation during imagery. He claimed that most imagery studies only showed activation in the latter posterior cortex areas (such as the parietal cortex and inferior temporal lobe) rather than in the visual cortex. He claimed that for the few imagery studies showing activation on the visual cortex, none presented conclusive evidence of a topographically mapping. Although Pylyshyn conceded that reasoning using imagery is different from logical reasoning, he concluded that "spatial displays" (i.e. visual depictive representations) are inadequate for the representation of knowledge.

## 2.4 Discussion

Figure 2-7 summarizes the positions between the two camps. As the last column indicates, the theorists explain the experimental results as being either architectural mechanisms or knowledge. In order to design a general, computational system incorporating imagery capabilities, one must make a commitment as to whether imagery processing requires specific architectural mechanisms or can be realized with general, symbolic computations and knowledge. Again, our hypothesis is that spatial and visual imagery use at least three distinct architectural representations. We back this assertion with the following arguments and with our evaluation in Chapter 7.

| Imagery Theorist | Representations | Behavioral Experiments | Neuroimaging Experiments | Explanation |
|---|---|---|---|---|
| *Depictive* | Symbols, Depictions | Response times linear with distance of transformation | Visual cortex active during visual imagery tasks | Architecture |
| *Propositions* | Symbols only | No correlation between response times and distance between objects | None, argue that most neuroimaging experiments do not activate visual cortex or show topographical map | Knowledge |

**Figure 2-7: Summary of Mental Imagery Debate**

First, Kosslyn and others propose a cohesive and consistent theory. Propositional theorists do not offer any compelling, competing theories to embrace their viewpoint and cannot always explain the major phenomena (e.g. rotation) without resorting to ad hoc arguments. Even they agree that some of their theories require excessive computations (Pylyshyn, 2002). Even though the propositional theorists disagree with the neurological evidence, they do not provide alternative explanations as to why the visual cortex is activated during some imagery experiments. If visual imagery is truly using only higher, amodal symbols, then why is there any activity in the visual cortex?

Second, we agree with Pylyshyn's argument that tacit knowledge explains some imagery results as behavior emerges from a combination of the environment, knowledge, and the architecture. However, we disagree that this explanation implicates propositions as the exclusive structure for imagery processing. In addition to the "island map" experiment, there have been many other behavioral experiments providing evidence that visual imagery representations use space. These experiments include image transformations, image size, and visual angle (Kosslyn, 1980). As further evidence neuropsychologists, such as Farah, Soso, and Dasheiff (1992) have shown that there is a visual field of view in both perception and imagery when, for medical reasons, they had to remove a patient's occipital lobe from one cerebral hemisphere. After removal, they found that the horizontal visual angle was reduced in half for both perception and visual imagery. However, the vertical visual angle remained intact. Tacit knowledge alone cannot explain this result.

Third, one of the shortcomings in the mental imagery debate, and possibly a cause of confusion in the interpretation of whether knowledge, architecture, or a combination explains an experimental result, is that the focus of the theorists' arguments is on visual imagery. Spatial imagery receives less attention yet there appears to be distinct brain structures, such as the parietal cortex, that are active during such tasks (Mellet et al., 2000). As Grush (2004) articulates, a reason for the confusion is that spatial imagery does not fit either the propositional or depictive metaphors. As with propositions, spatial imagery representations are sentential and consist of objects with properties such as direction, distance, size, and motion. However, transformations between states does not follow logic or entailment but rather mathematical (e.g. translation, rotation, scaling) or dynamic (e.g. force, torque) manipulations. On the other hand, it is not a depiction either, such as an image or topographically organized visual cortex. The distinction between the two representations is difficult to appreciate because one can reinterpret the spatial representation into a depictive format. For example, a line can be represented in its algebraic, sentential format (y = x) or a depictive format (Figure 2-8).

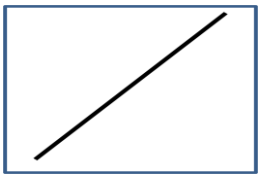


**Figure 2-8: The Depictive Format of the Line, y=x**

We argue that the difference in the results between Pylyshyn's and Kosslyn's island map experiments are attributed to the type of imagery task (spatial versus visual). Pylyshyn's version of the island map experiment is clearly a spatial reasoning task as the subjects were to determine the absolute, cardinal direction between two objects. Our theory offers the following explanation. As the subjects were memorizing the map, they encoded the qualitative directions between the pairs of objects (e.g., the well is left-of the tree, see Figure 2-4) and used this information to infer the cardinal direction. Note that encoding these local spatial relationships facilitates rebuilding a depictive representation, but it is not required to complete this task.

In Kosslyn's version of the experiment, the task was to "scan" between the two objects. Therefore, even if the subjects had tacit knowledge that it takes longer to scan

between objects further apart, the subjects have to account for the distance between the objects. As it is unclear whether this is a spatial and/or visual task (individual preferences may be a factor in this determination), our theory offers two explanations. Either the subject, using a previously encoded or measured distance, builds a quantitative spatial representation and "simulates" scanning between the two objects, or the subject generates a depictive representation and scans between the two objects by imagining a "path." In either case, perceptual-based representations and processes are in use.

Our final thoughts are that depictive theorists, such as Kosslyn, have not denied that there are symbolic computations involved in imagery processing. On the contrary, he specifies how symbolic, associative memories are required to build or generate an image. Past research with Soar has focused almost exclusively on symbolic computations. However, as Newell (1990) stated in *Unified Theory of Cognition*, imagery may be a component with a different representation existing outside of central cognition. We have pushed Soar to its limits using symbolic computations. With mental imagery as our motivation, it is time to explore others.

# Chapter 3

# Design Space Constraints and Theory

As discussed in the previous chapter, the use of multiple representations is a core constraint of our theory. In this chapter, we discuss the remaining core constraints influencing the architectural design space. Our theoretical commitments closely follow the evidence provided by the depictive imagery theorists, specifically Kosslyn (1980; Kosslyn, 1994; Kosslyn, Thompson, & Ganis, 2006).

Figure 3-1 categorizes our design space constraints into three areas: behavioral/biological, functional, and computational. We derive behavioral and biological constraints from a literature review of the theory and mental imagery experiments measuring behavioral or neural responses through neuroimaging techniques (i.e. PET, fMRI). Appendix A summarizes the notable experiments influencing our theory. Functional constraints emerge from the behavioral and biological constraints so there is some overlap. As we are extending the Soar cognitive architecture (Laird, 2008), we derive the last three computational constraints from Soar's computational model (Newell, 1990). The integration of the functional constraints with a cognitive architecture and the computational support for efficient processing of the representations are the major contributions of this research.

| Behavioral/ Biological | Functional | Computational |
|---|---|---|
| • Integrate and use multiple representations<br>• Share mechanisms with vision<br>• Organize by parts<br>• Composability<br>• Transformation<br>• Inspection / "Visualize" | • *Quantitative spatial and visual depictive structures*<br>• *Construction*<br>• *Transformation*<br>• *Generation*<br>• *Inspection*<br>• *Maintenance* | • *Efficient processing of the representation*<br>• Clear separation of knowledge and architecture<br>• Support reactive and deliberate behavior<br>• Problem space computational model |

**Figure 3-1: Design Space Constraints**

## 3.1 Behavioral and Biological Constraints

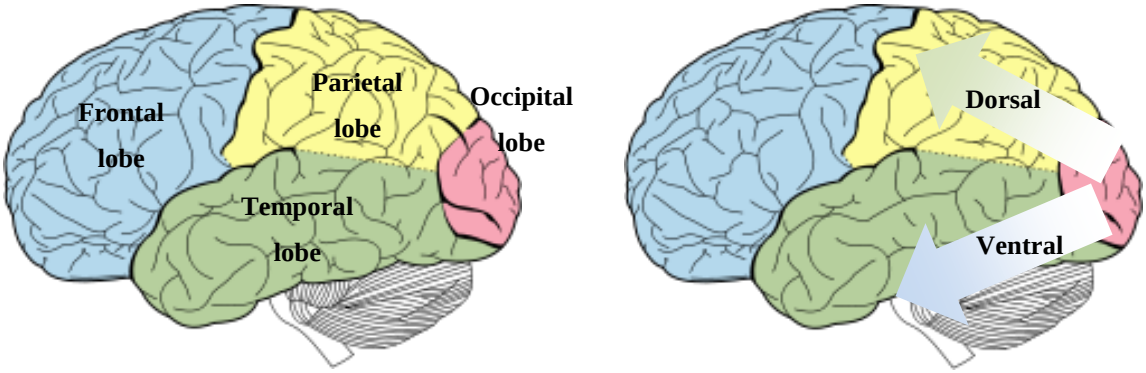
There are many studies showing that vision and imagery share similar characteristics to include visual and spatial structure, resolution limits, field of view, laws of motion dynamics, motion aftereffects, short-term and long-term memories, and interference patterns (Farah, Soso, & Dasheiff, 1992; Finke, 1989; Gilden, Blake, & Hurst, 1995; Kosslyn, 1980; Palmer, 1999; Peronnet, Farah, & Gonon, 1988). Finke (1989) calls this the *perceptual equivalence principle*.

> "Imagery is functionally equivalent to perception to the extent that similar mechanisms in the visual system are activated when objects or events are imagined as when the same objects or events are actually perceived."

The primary difference between vision and imagery is the source of information (i.e. retinal input versus memory activation) and the initiation of the processing (i.e. top-down versus bottom-up).[5] Imagery may build spatial and visual representations entirely from the combination of activated object and spatial memories or by augmenting a perceived scene with objects and their spatial configurations from these memories. Thus, the interpretation of an imagined representation can occur in the presence or absence of perceived stimulus. This design space constraint requires that spatial and visual imagery share components associated with vision rather than having their own, separate mechanisms.

[5] However, most present theories of visual perception include substantial, "top-down" processing that employs knowledge about objects to facilitate segmentation, recognition, and classification. Spatial and visual imagery are considered part of this top-down processing.

Neurological evidence shows imagery's integration with vision begins at the visual cortex. The visual cortex is the region of the occipital lobe (Figure 3-2a) that processes visual information received directly from the lateral geniculate nucleus (LGN). The LGN, in turn, receives information from the retina. From a biological perspective, the visual cortex is the "lowest" area of the brain where imagery experiments have shown activation. From a functional perspective, this area is associated with the visual depictive representation.




**(a) Human brain lobes** **(b) Ventral ("what") and dorsal ("where") pathways**

**Figure 3-2: Shared Biological Mechanisms between Imagery and Vision**

Two pathways emanate from the visual cortex (Ungerleider & Mishkin, 1982). The ventral, or "what" pathway, extends from the visual cortex to the inferior temporal lobe while the dorsal, or "where" pathway, runs from the visual cortex to the posterior parietal lobe (Figure 3-2b). The ventral pathway includes processes that extract an object's visual features and attempt to recognize the object by matching the features to an object in a long-term memory (Kosslyn, 1994; Palmer, 1999; Ullman, 1996). This long-term object memory, assumed to be in the inferior temporal lobe, encodes the shape and color of objects. Some researchers advocate a 3D model (Finke, 1989; Marr, 1982; Pinker, 1988). Others suggest a population code (Kosslyn, Thompson, & Ganis, 2006). We are non-committal in this regard, as it remains unclear from our research how information in this memory is encoded.

What is clear is that the system is able to reacquire shape and color. The shape may be a prototypical representation of the object (e.g. a prototypical chair), or in some cases, where one is exposed to the object through multiple repetitions, a very specific

shape that represents the exact object (e.g. the chair in my dining room) (Weaver, 1993). Although the shape and color representation in this memory is unique, our theory does not address it as a separate representation as it is not directly used in reasoning (i.e. there are no processes that directly manipulate it). Rather, the representation is instantiated during imagery to support constructing the quantitative spatial or generating the visual depictive structure.

As the processes along the ventral pathway are extracting visual object features, the dorsal pathway processes are extracting spatial properties from the visual cortex, such as an object's location, orientation and size, and transmitting this information to a short-term spatial memory in the parietal lobe (Kosslyn, 1994; Palmer, 1999). This short-term spatial memory is associated with quantitative spatial representation. During perception, it is an egocentric representation (i.e. relative to the head direction), and during imagery, the representation can be either from an egocentric or allocentric viewpoint. Long-term memory for spatial representations are encoded as allocentric representations in the medial temporal lobe (Byrne, Becker, & Burgess, 2007).

One of the problems with the "what" and "where" analogy is that there appears to be little understanding on how perceived objects are reconciled between the two pathways. That is, how do ventral path processes associate objects they are recognizing with objects from which the dorsal path processes are deriving spatial properties? Pylyshyn (2001) offers some insight here. In addition to a set of "what" and "where" processes, there seems to be a "which" process responsible for indexing and tracking objects in the perceived scene even though their location and properties change. Pylyshyn calls his theory, *visual indexing*. As an analogy he compares it to a demonstrative in natural language, such as "*That* is red," where "that" is the visual index we picked out from our visual field.

In Pylyshyn's theory, there is a preprocessing phase where a process selects and indexes a few objects (4-5) in the scene. This phase is distinct and precedes object and spatial recognition. How the preprocessing selects the salient objects is beyond the scope of Pylyshyn's theory, but Itti (2000), Marr (1982), and Ullman (1996) suggest that contour, color, motion, and orientation patterns from the depictive representation contribute to the determination of salient objects. Pylyshn states that only indexed visual

objects enter subsequent processing and argues that such an indexing scheme facilitates recognition and tracking the objects. This "binding" issue also has ramifications for imagery. That is, in order to inspect the features of a specific object or a spatial relationship between two objects there has to be an index, or referent, to the object(s) in question.

Once we have adequately understood the visual and spatial memories and processes perception uses, we can begin to understand how imagery leverages these mechanisms. A commonly demonstrated phenomenon in behavioral imagery experiments is that the time to generate an image is linearly dependent on the number of parts, or objects, in the representation. The construction of mental images arises from the amalgamation of metric shape and descriptive, symbolic knowledge. The ease of visualizing an object is dependent on the number of parts composing the object and how the parts are arranged in the symbolic description (Finke, 1989; Kosslyn, 1980, 1994; Kosslyn et al., 1983; Kosslyn, Thompson, & Ganis, 2006).

Another common behavioral phenomena, made famous by Shepard's and Metzler's (1971) "mental rotation" experiment, is the ability to imagine the transformation of objects in a scene. One can change either their viewpoint from egocentric to allocentric or translate, rotate, or resize imagined objects. Finke (1989) calls this the *transformational equivalence principle*.

> "Imagined transformations and physical transformations exhibit corresponding dynamic characteristics and are governed by the same laws of motion."

## 3.2 Functional Constraints

The functional constraints emerge from the behavioral constraints. The architecture must account for how imagery processing *constructs*, *transforms*, *generates*, *inspects*, and *maintains* the spatial and visual representations. These functional constraints must show through the architecture in the following ways. First, the descriptive representations of the objects, or parts, must be organized in a compositional manner. Objects may be composed of other objects ("has-a" relationship), which in turn may be composed of more primitive objects. For example, a village is composed of buildings and roads.

Buildings in turn are composed of many rooms, which are composed of chairs, tables, beds, etc. *Local s*patial relationships between objects and their parts, such as the chair's arm is above and to the right of the seat, are similarly organized. This organization occurs when the information is stored during perception rather than when it is retrieved from memory and is a result of the temporal or spatial sequence from which it was originally perceived in the environment (Kaplan & Kaplan, 1982).

Second, imagery is an incremental addition or deletion of objects or shapes. It may involve a novel combination of objects (e.g. an elephant on top of a house), a previously seen object or scene (e.g. my living room), or novel patterns (e.g. imagining a path on a map) (Kosslyn, 1994). Object information, such as its shape and color originate from the long-term object memory. Spatial properties, specifying the location and orientation between objects, are activated from a declarative long-term memory and can be qualitative (i.e. left-of, above, disconnected) or quantitative. If in a qualitative format, the system must interpret the qualitative representation and convert it to a quantitative representation. In such cases the specification may be under constrained (e.g. imagine "A" to the left of "B") and open to multiple interpretations. In these cases, task knowledge or default heuristics, such as the notion of an *influence area* (Kettani & Moulin, 1999) can define the distance between objects.

The third way the architecture must reflect these functional constraints is to support the transformation, or manipulation, of a quantitative spatial or visual depictive representation. The quantitative spatial representation and associated processes must provide the ability to modify the viewpoint or change the location, orientation, or size of one or more objects in the scene. Manipulating visual depictive representations may be with mathematical processing (e.g. rotation, scaling, and filters) or algorithms that take explicit advantage of the topological structure and color.

The fourth functional constraint, generation of a visual depictive representation, requires the architecture to provide mechanisms to render a scene from a privileged viewpoint. Again, rendering must be efficient and support the acquisition of an image. In Chapter 6 we will discuss how the transformation and generation constraints together influence our choice to use a *scene graph* for the quantitative spatial representation.

The ability to "visualize," or inspect, a quantitative spatial or visual depictive representation to infer spatial or visual properties reflects the primary purpose for spatial and visual imagery. Imagery does not have its own set of inspectors, or feature and spatial detectors. It simply relies on those mechanisms it shares with vision. Therefore, after the system constructs and, if necessary, transforms and generates the image, the flow of information proceeds as in bottom-up perception.

Finally, since visual imagery and visual perception coexist, sharing the same region of the visual cortex, the architecture must support maintenance of the visual depictive representation. Our resulting architecture does not support image maintenance in the sense that the visual depictive representation must continually be "refreshed" or it begins to fade (Kosslyn, Thompson, & Ganis, 2006). However, we do include architectural mechanisms that inhibit additional incoming stimuli from disrupting the focused representation. Otherwise, perception always trumps imagery, never allowing it to finish.

## 3.3 Computational Constraints

A cognitive architecture is the fixed set of memories and processes underlying an agent (Newell, 1990). The motivation behind a cognitive architecture is that with the addition of knowledge, it can support intelligent behavior across a wide variety of tasks and environments. The architecture must support knowledge acquisition through perception and learning and provide mechanisms to encode, store, retrieve, and process the knowledge to enable planning, coordinating, and executing actions in the world.

Cognitive architectures have traditionally represented knowledge as symbolic (e.g., rules, semantic nets, frames) structures. Perceptual-based representations have received less attention as a form of knowledge representation. One of our motivations, reflected by our first two computational constraints (Figure 3-1), is the possibility that an agent can achieve a computational gain using perceptual-based representations while maintaining clear separation between knowledge and the architecture.[6] As we will

[6] In a depictive representation, some of the "knowledge" is tacitly in the architecture -- the grid geometry embeds knowledge of the plane that would have to be explicitly encoded in a purely symbolic system. That is one of the strengths of the depictive representation.

demonstrate through our architecture description and evaluation, specialized, architectural components processing these representations can achieve an order of magnitude (or more) gain over symbolic processing without trading off generality.

Supporting reactive and deliberate behavior, our third computational constraint, is a hallmark of Soar and one that we wish to maintain with the addition of spatial and visual imagery. That is, the computations that the imagery subsystem performs must meet practical computational requirements. The computational cost of a single "step" of building, transforming, generating, or inspecting a spatial or visual representation should work within the time constraints of one Soar decision cycle that is hypothesized to be 50 milliseconds in humans. Otherwise, the system is not responsive to changes in the environment. Note that this constraint is different from the first computational constraint (support efficient processing of the representation). A specialized perceptual process could be more efficient relative to performing the same operation with symbolic computations, but not responsive. For example, one of our implemented feature detectors for identifying curves in a depictive image is much more efficient than trying to detect curves using symbolic computations (in fact, we are not sure if it is even possible). However, it requires too many computational cycles (~6 seconds real CPU time) to be considered reactive to the environment. This shortfall may be a result of the wrong choice of an algorithm, poor implementation, or non-parallel hardware, but as it is currently implemented, it is not reactive and thus violates this constraint.

The final computational constraint (Figure 3-1), the problem space computational model (PSCM), provides the control constraint necessary for imagery processing. PSCM is a paradigm for realizing intelligent behavior and is the basis for Soar (Newell et al., 1991). A problem space consists of a set of *states* and a set of *operators*. An agent, by iteratively selecting and applying operators, effectively conducts a search through its problem space. During each cycle, the agent executes a *knowledge search* to bring all the relevant knowledge to bear in deciding what operator to choose next.

In this computational model, imagery is a special problem space using specialized mechanisms for general spatial and visual processing. The architecture maintains control

with operators for constructing, transforming, generating, and inspecting a quantitative spatial or visual depictive representation. During each imagery cycle, the agent conducts a knowledge search of its memories to build a quantitative spatial representation, generate a visual depictive representation, or transform or inspect either representation to facilitate further reasoning. The imagery process is conditional and iterative. The agent may add more detail to its representation(s) and inspect it to refine the search.

## 3.4 Theory Summary

Figure 3-3 summarizes the theory. What Newell describes as central cognition encodes knowledge in the form of amodal, symbolic representations (Newell, 1990). Some of the knowledge is a representation of objects in the world or *visual objects*. During perception or memory retrieval, the architecture creates *visual symbols* representing these visual objects. The visual symbols denote either a prototypical object (e.g. a chair), a specific instance of an object (e.g. the chair in my dining room), or multiple objects (e.g. my dining room).

Imagery is the combination of the imagery problem space and the specialized imagery subsystem. The agent initiates the imagery problem space when there is an impasse in problem solving and wants to resolve the impasse using spatial or visual imagery. Task operators direct the imagery problem space that in turn controls and communicates with the imagery subsystem through its operators (*construct, transform, generate, inspect).*

The *construct* operator triggers the construction of a quantitative spatial representation. The imagery subsystem builds the structure by combining the general, metric shape of objects from a long-term object memory with qualitative or quantitative spatial information from a symbolic memory and encodes the resulting representation in a spatial short-term memory (STM). The *transform* operator controls the transformation of the spatial representation by manipulating its viewpoint or specific objects within it. An imagery *generate* operator creates a visual depictive representation in a visual STM by combining the quantitative spatial representation with each object's specific shape and color from the object LTM and renders it from a specified viewpoint. A *transform* operator manipulates the depictive representation by activating specific regions. During

inspection, perceptual visual and spatial processes perform reasoning by searching the visual STM or spatial STM for visual and/or spatial properties. The results of the inspection are transmitted to central cognition where the imagery problem space operators build the internal symbolic memories. Central cognition uses the inspection results to continue progress through the current task's problem space.

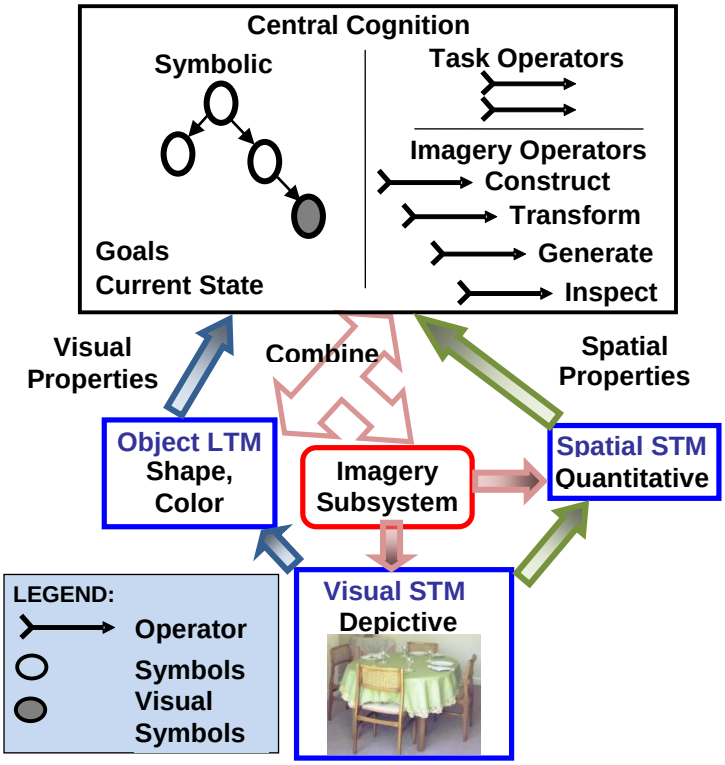



**Figure 3-3: Summary of Spatial and Visual Imagery Theory**

Note that although we have assigned the symbolic representation to central cognition's associative memories, the quantitative spatial representation to the spatial STM, and the visual depictive representation to the visual STM, this is not to claim that each memory contains that type of representation exclusively. Symbolic memories in central cognition may have quantitative representations; the spatial and visual memories may contain symbols, and so forth. What we do claim, however, is that each memory has specialized *processing* mechanisms for their primary representation, and these mechanisms are what distinguishes the memories. Therefore, although a symbolic computation in central cognition may be able to process a quantitative or depictive representation, it cannot do so as efficiently. Although it is not an established part of our

theory or resulting architecture, what it does indicate is that there is some redundancy so if one component is resource-constrained or incapacitated, alternate memories and processes can assist.

In summary, decision-making proceeds by combining perceptual representations with task specific knowledge to construct an imagined scene. Analysis emerges through the manipulation of both sentential and depictive representations. Retrieval or inspection of the resulting representations provides new information that the agent uses to reason and produce action in the environment. We will reiterate how these design space constraints influence the architectural design in Chapter 6. First, however, we turn to describe previous computational approaches.

# Chapter 4

# Related Work

## 4.1 Cognitive Architectures

Although cognitive science and artificial intelligence (AI) researchers have made enormous progress in peripheral disciplines, there has not been a previous effort to support spatial and visual imagery processing within a cognitive architecture. Until recently, architectures such as Soar (Laird, 2008) and ACT-R (Anderson et al., 2004) focused on higher-level cognition, or what Newell calls central cognition (Newell, 1990) and typically ignored perceptual and motor mechanisms. There is strong evidence, however, that the environment plays a key role in cognitive processing and the perceptual and motor systems serve as the link that integrates the environment to higher-level cognition (Barsalou, 1999; Kaplan & Kaplan, 1982) .

From its inception, the EPIC architecture (Kieras & Meyer, 1997) emphasized the perceptual and motor systems. However, rather than specifying and implementing the low-level details of perception and motor processing, (e.g. edge detection, joint coordinates), EPIC focuses on temporal constraints between perception, motor, and cognitive components to account for human dual-task performance. Perception provides symbolic input to cognition and cognition sends symbolic output to the motor system. There are no quantitative or depictive representations involved in the reasoning.

Following EPIC's lead, Soar and ACT-R extended their architecture to include integrated perceptual and motor systems. EPIC-Soar (Chong & Laird, 1997) integrated EPIC's perceptual and motor processor modules with Soar to evaluate the performance and acquisition of executive process knowledge that is required to support the execution of two concurrent tasks (i.e. dual-task). The architecture was similar to our approach in the sense that EPIC provided perceptual input to Soar, and Soar sent motor commands to EPIC. However, the EPIC-Soar hybrid architecture was two independent processes and not integrated where one component can take advantage of another's mechanisms.

The current version of Soar takes a more functional approach, using an external module that translates an environment's perceptual information into a symbolic representation Soar can use for reasoning. Likewise, a module external to Soar, transforms the symbols Soar sends from its working memory into a format that produces behavior in the environment. Using these perception and motor modules, Soar has demonstrated success in modeling human behavior in dynamic environments to include military simulations with pilots flying fixed-wing or rotary-wing aircraft and soldiers conducting Military Operations on Urban Terrain (MOUT) (Jones et al., 1999; Tambe et al., 1995; Wray et al., 2005). Again, however, these modules do not perform any type of cognitive functions.

Similar to EPIC, ACT-R's perception and motor modules focus more on the timing and content of modalities rather than the representational format and low-level processing capability (Anderson et al., 2004). In the case of perception, productions request the visual module for information based on constraints. For example, a production may request the "red" object or the object that is located "on-top" of the current scene. One of ACT-R's important extensions to EPIC's model of the visual system includes breaking visual perception into two modules each with a short-term memory (buffers in ACT-R terminology). The visual-object, or "what," module holds the symbolic features of the object currently being attended to by the visual-location, or "where," module. The visual-location module maintains the location of all the objects in the current scene. However, there is not long-term perceptual memory enabling the persistence of an object's shape or color and no short-term memories for manipulating perceptual representations and drawing inferences from them.

Gunzelmann and Lyon (2007) have recently proposed an extension to ACT-R that includes a module specialized for spatial information processing. The proposed module contains processes to perform both qualitative and quantitative, mathematical comparisons of spatial relationships between objects, such as direction and distance. They recommend that the spatial module have connections to the visual and motor modules (contrary to ACT-R theory of no direct connections between modules) for the purpose of extracting spatial properties from perceptual input and executing motor control. Their proposal appears to be similar to our theory of spatial imagery and corresponding implementation. Their proposal does not include plans for incorporating depictive representations and processing.

Wintermute's and Laird's (2007) Soar Spatial Reasoning (SRS) system focuses on how the architecture projects qualitative predicates into a quantitative spatial representation, providing a more detailed implementation of the capability than our current system provides. Kurup and Chandrasekaran (2007) have also argued for multi-modal architectures and augment Soar with their diagrammatic reasoning system. We will discuss the similarities and differences between our approaches shortly.

There have been several other efforts to extend the perception and motor capabilities of each of these architectures (Hill, 1999; Hill, Han, & Van Lent, 2002; St. Amant et al., 2005). Each contribution effectively pushes the architecture closer to the environment. The problem with these approaches, however, is that they assume the cognitive system abandons the perceptual representations rather than using them to participate in problem solving. Discarding these representations adversely affects the system's ability to perform visual and spatial reasoning and requires ad-hoc, bolted-on components that are tailored for specific domains (Best, Lebiere, & Scarpinatto, 2002; Wray et al., 2005). What we are missing from these architectures is the ability to amalgamate the cognitive and perceptual representations in a general-purpose way and then use the resulting information for reasoning.

## 4.2 AI Systems

There have been several AI diagrammatic reasoning systems built that use both symbolic and quantitative representations. Gelernter's (1959) geometry theorem proving machine

is perhaps the earliest. Kurup's and Chandrasekaran's (2007) biSoar (discussed below) is the most recent and is the closest parallel with our work. With the exception of Funt's, (1976) WHISPER system, there have been few systems that reason with depictive representations. Funt argued that people solve problems on four levels, and AI was ignoring the last level. The levels were (1) the goal-oriented approach where the system searches for a solution; (2) the mathematical level where sentential equations enable progress; (3) the relational level where a complete structure of the explored search space is used for reasoning; and (4) the image level where representations were analogues of the situation. Funt argued that depictive representations overcame the "frame problem" because objects move together so the system does not have to use computational cycles to infer new features and relationships. Empty space and new shape just emerge. Despite Funt's seemingly convincing arguments, the mainstream AI community has continued to ignore the use of depictive representations as a form of knowledge representation and reasoning.

The specific problem WHISPER solved was determining the stability of a stack of arbitrarily shaped rigid bodies. WHISPER consisted of symbolic qualitative physics rules ("if a block is on a slant, it will slide"), an image of the situation, encoded as a two-dimensional array, and basic algorithms for modifying the image. The rules directed a simulated parallel processing "retina" capable of extracting basic, domain independent features (e.g. object contact, object symmetry, finding the center area of an object). Each unit in the retina was constrained to communicate only with its immediate neighbors and a "retinal supervisor" that consolidated each unit's inspection results for a particular query. The local communication constraints between retinal units resulted in algorithms similar in spirit to Furnas' (1990; Furnas, 1991; Furnas et al., 2000) pixel rewrite system that we use as motivation for some of our depictive processing.

Marr (1982) addressed many of the underlying issues of how the visual system recognizes object features in a scene with his seminal work in computational vision His work influences our design space in two important ways. First, we apply his notion that visual processing produces incremental, increasingly abstract levels of representations (i.e. the pixel image, raw primal sketch, 2 ½ D sketch, 3D model, symbols). In a similar manner, but in the opposite direction, imagery starts with the symbolic representation,

combines it with stored perceptual memories to produce increasingly concrete representations. Marr stressed how certain formats made certain types of information explicit and accessible. From the perspective of top-down cognitive processing, we can also make this argument,—which is a primary reason for pursuing quantitative spatial and visual depictive representations in problem solving. If certain representations are useful for extracting features such as surface contours, object orientations, or spatial relationships in bottom up visual perception, then these representations are also useful in top-down processing when the information is not explicitly encoded as symbols. This principle is in accordance with Newell's theory that intelligent systems should bring all knowledge to bear in solving a problem (Newell, 1990).

Second, Marr's theories concluded that visual perception stores an object's 3D model, so it can recreate its shape if required. An important part of our architectural assumptions is that we assume the system does not just throw this shape information away after it recognizes the object. At a minimum, it must be encoded for subsequent recognition. In the case of spatial and visual imagery, it is activated to support further reasoning.

Tolman (1948) articulated how rats represented spatial knowledge, or *cognitive maps*, to assist them in finding food in a maze. The cognitive map metaphor, or the representation of large-scale space, provide a psychological theory of how we acquire an object's location, orientation, and size relative to other objects in the currently perceived scene (Kaplan & Kaplan, 1982). The theory also specifies how we connect individual scenes together. Cognitive map theory says that as you explore the world, you begin building up representations of the relationships between static objects or landmarks in the environment and the relationships between objects in individual scenes. Cognitive maps provide important concepts for spatial imagery because they provide a starting point as to how spatial knowledge is organized. Spatial imagery can rebuild scenes by composing objects together using the same *local* spatial relationships derived from cognitive maps. This provides a methodology for reconstructing representations of previously seen objects to infer new *global* spatial relationships. Cognitive map theory also advocates the idea that there is a *viewpoint* associated with the stored spatial relationships. Some have

hypothesized that these maps are stored from a first-person, egocentric viewpoint with gateways, or exits, separating the scenes (Chown, Kaplan, & Kortenkamp, 1995).

Cognitive map theory differs from spatial and visual imagery in that it does not address how specific knowledge about an object (i.e. shape) is stored or later retrieved. The theory also does not address how task knowledge, information from other modalities, or dynamic objects are combined to form new spatial representations. For example, using cognitive map theory, you can recall the main intersection in the center of a small town. Your scene may contain the roads, buildings, and traffic signs but does not include dynamic objects, such as cars or people walking. The scene is "remembered" from one particular vantage point. Spatial imagery enables one to imagine the scene, add dynamic objects to it, transform either the viewpoint or specific objects, and then inspect the scene for specific knowledge.

Kuipers, leveraging the cognitive map metaphor, developed the Spatial Semantic Hierarchy (SSH) with the goal of explaining how a robot learns the spatial structure of the environment. Each hierarchical layer has qualitative and quantitative representations with global knowledge of the environment increasing as you move up the hierarchy from very specific control laws to topological maps of places, paths, and regions. At the highest level, or what Kuipers calls the global metrical map, SSH combines the qualitative (symbolic) topological relationships with the local, two-dimensional geometry to form a structure with one global, allocentric frame of reference. Whereas Kuipers focuses on how spatial structures are acquired, we concentrate on how the spatial structures are used in general problem solving.

The closest parallel to our work, is that of Kurup (2008) and Chandrasekaran (Kurup & Chandrasekaran, 2007). Their system, biSoar, combines the Soar cognitive architecture with their diagrammatic reasoning system (DRS) and reasons using both symbolic and diagrammatic representations. Similar to imagery construction, transformation, and inspection, their system has a set of *action routines* to add diagrammatic elements and *perceptual routines* to extract spatial relationships from the diagram.

There are a few key theoretical and implementation differences between our approaches, perhaps because they have focused more on diagrammatic reasoning and we

have considered the psychological and neurological constraints of imagery. First, they propose a single, working memory containing both symbolic and diagrammatic representations while we advocate separate symbolic and perceptual memories where the symbolic, procedural memory does not have direct access to perceptual-based representations. We base our decision on evidence that modality-specific representations (i.e. spatial, visual, auditory) are distinct posterior neural systems (see (Jonides et al., 2008) for a review of working memory theories). From a computational standpoint, a primary reason for having a multi-representational system is to gain a computational advantage using processes that are specific to the representation. By embedding the perceptual representation into a symbolic computational system, you lose this efficiency without mechanisms to distinguish the two. Although the two implementations are similar (i.e. their diagrammatic reasoning system is outside of Soar), Kurup and Chandrasekaran base their theory on the notion that by having a single, multimodal working memory, automatic learning of both symbolic and diagrammatic representations can occur using Soar's chunking mechanism. We currently do not have such a theory of how results from imagery processing are learned (except, perhaps as an encoded episode), and are not clear as to how such a theory would be realized from a practical standpoint.

Second, their diagrammatic reasoning theory specifies the type of objects (point, curve, and region) a diagram can contain while we leave the type of object open-ended to any shape and color the agent experiences in the world, imagines by composing known objects, or emerges from the manipulation of a depiction (i.e. a new shape). Our approach leaves the complexity, detail, and richness of an imagined scene much more open where, in addition to specifying distance, direction, and topology our representations and processes also consider the orientation (i.e. front), specific shape, and color of an object.

Finally, they are noncommittal as to whether diagrams are quantitative, algebraic equations or depictive, image representations. Their current implementation uses sentential, metric structures. For example, points are two-dimensional, Cartesian coordinates, lines are composed from two points, a curve is a sequence of straight lines, and a region is a closed curve. We make a distinction between the two representations, as there are different types of reasoning that can be performed on each (e.g. extracting visual

features from a depictive representation). In the end, however, we are both motivated by how a task-independent architecture uses amodal symbolic and perceptual representations in reasoning.

## 4.3 Computational Models

Previous efforts to build computational models of either spatial or visual imagery have not included the constraints of a general cognitive architecture. Kosslyn (1980) composed a detailed computational model of visual, depictive imagery. Although the model clarified his theories, Kosslyn did not build it with the intent of incorporating it into a cognitive architecture.

Baylor (1971) implemented a computational model of the block visualization task.[7] What is interesting about his approach is that he divided the knowledge into two problem spaces. The symbolic space manipulated generic information about blocks, and the image space (implemented with symbolic representations) had specific operators that manipulated visual information. This problem space division is similar to our computational theory.

Moran built a computational model of spatial imagery using a production system (Moran, 1973). The task he chose to model began with the agent at a specific location. The agent is then issued a series of directions (e.g. move north one-step, turn east, move forward one-step, etc.) and is to "report" its final location and direction. Moran raised some valid points such as how these representations are constructed and controlled. However, we disagree with his hypothesis that imagery is *entirely* symbolic in nature. Since the task was spatial, rather than visual, depictive representations were not required. Moran argued that pictorial representations are uneconomical, as they require a large amount of information to be stored. We agree that recording every scene would quickly exceed our memory capacity. That is why an object long-term memory only stores compact shape and color representations. It is the task of imagery to recreate the picture.

[7]The task requires a subject to start by visualizing a three-inch cube and imagine one of its side's red. Next, the experimenter instructs the subject to imagine two sides blue. The blue sides are adjacent to the red side but opposite each other. Finally, the subject imagines breaking the cube into one-inch cubes and deciding how many of the resulting cubes have exactly one red and one blue face.

Glasgow and Papadias (1992) built a molecular scene analysis application using mental imagery as motivation. Their system uses three separate representations (descriptive, spatial, visual) and the visual representation (occupancy array) is similar to our depictive representation with the exception that they only render convex shapes. Their long-term memory, where they store descriptive representations is similar to our symbolic representations in Soar. There are, however, three major differences. First, they built a specific application while we are taking a more general approach. While Glasgow and Papadias took significant strides to incorporate key findings in mental imagery, they did not design it with the overarching constraint of a cognitive architecture. Second, they represent spatial information using symbolic arrays rather than a quantitative format. Finally, they make no commitment as to how the visual representation is constructed, where the shape information is stored, or how more than one object is arranged in the visual representation.

Tabachneck-Schijf's et al. (1997) CaMeRa model uses multiple representations and simulates the cognitive and visual perceptual processes of an economics expert teaching the laws of supply and demand. Their system includes both visual short-term and long-term memories that complement verbal memories. Visual STM includes a quantitative (node-link structure) and a depictive (bitmap) representation that is similar in design, although not in implementation, to our representations. The architecture's overall generality is unclear although it appears to be their intention. Their shape representation is limited to algebraic shapes (i.e. points and lines) and their spatial structure only models an object's location while ignoring orientation and size.

Barkowsky's (2002) MIRAGE application relies on mental imagery evidence to reason about space in a geographic context. It focuses primarily on how mental images are constructed from qualitative geographic spatial relationships. Barkowsky (in press) proposes that any model of mental imagery must include the following:

(1) Hybrid representational formats to include propositional and visual structures involving shape.

(2) Coupling between imagery and visual perception.

(3) Construction of images from pieces of knowledge.

(4) Processing with or without external stimuli.

(5) Multi-directional distributed processing and control.

Our architecture addresses (1) – (4). Our control structure initiates imagery processes in a top-down manner while perceptual mechanisms process results in a bottom-up fashion. In Soar, the contents of working memory determine which memories and processes are active without any centralized control (5). In addition to Barkowsky's list, we also propose that the architecture must support transformation and generation of a depictive representation.

# Chapter 5

# Tasks and Environments

When does an agent use spatial and visual imagery? How does it know which representation to use? This chapter begins by summarizing characteristics of the tasks and environments where spatial and visual imagery is useful. The characteristics include a discussion and examples of general tasks and specific sub-tasks requiring spatial or visual imagery. We then provide three concrete examples by introducing the tasks and environments we use to evaluate the architecture. The first two tasks are primarily internal problem-solving tasks where there is limited interaction with an external environment. The final task extends the first two tasks to a dynamic and continuous environment where the agent must interpret and act upon information from multiple sources and perception and imagery must interact and share the same resources.

## 5.1 Characteristics of Tasks and Environments

In general, spatial and visual imagery is useful in tasks requiring the inference of spatial relationships (direction, distance, orientation, topology, size) between two or more objects or detection of an individual object's spatial (e.g. width, height, orientation) or visual (e.g. shape, color) properties. In both cases, imagery is useful because the spatial or visual information required to make a decision is not directly accessible from either perceptual input or memory retrieval. These situations include circumstances where vision would normally perform the analysis, but the relevant objects or spatial

configurations are hypothetical, missing details, or not present. However, by combining the information into a quantitative spatial or visual depictive representation, one can infer the relevant spatial or visual detail. The external environment may have many spatial and visual characteristics or none at all, but the task is such that imagining the situation helps clarify its spatial and/or visual properties.

As an example, consider a young child playing hide-and-go-seek with a parent inside their home. The parent, playing the role of the "hider" may provide audio clues to assist the child's search. The child, having inadequate perceptual input (he or she does not see the parent), may use spatial imagery to retrieve a stored representation of the spatial layout of each room and combine it with the parent's audio signal to guide the search. As the child "seeks," she may use visual imagery to focus in on specific locations where the parent might "fit." In this example, both the environment (rooms in house) and the task (hide-and-go-seek) have many spatial and visual characteristics. The rooms have direction, distance, and topological relationships, hiding places have size, and the task requires filling in the missing spatial and visual details (e.g. what direction should I move to next? Will daddy fit in the closet? In the cupboard?)

On the other hand, the immediate environment may not always include spatial or visual features. Consider when you read a story or someone is giving you verbal directions. In these examples, you may create an imagined scene to achieve a better understanding of the spatial and visual properties of the task. However, the surrounding characteristics of the environment are irrelevant. Even if the environment includes many spatial and visual features, imagery may not be useful if the task is repetitive and highly learned (e.g. driving to work).

We suggest there are four general tasks where using imagery is useful to infer spatial and visual properties.

- Filling in missing details of a situation
- Recognizing novel shapes and spatial properties if not present visually
- Analyzing or rehearsing an outcome of an action before executing the action
- Replay of a previous event to inform a future decision

The previous two examples (hide-and-go-seek, imagine a scene from a story) are instances of the first two tasks where perception provides none or only part of the spatial and visual properties required for reasoning. Our three experimental domains described below are also instances of these first two tasks. The third general task uses imagery as a simulator to analyze or rehearse the possible outcomes of *future* actions where simulating actions involve moving imagined objects or looking at the scene from a different perspective. We explore this general task in our third experiment. The last general task also uses imagery as a simulator, but rather than simulating potential future states, reasoning simulates a *previous* state or event to inform a future decision. This form of reasoning requires the retrieval of previous experiences from an episodic memory. We do not evaluate this task but will address it as part of our future work.

Each of these general tasks will include specific spatial and visual subtasks, some of which we describe next. Schultheis et al. (2007) have suggested the following criteria to distinguish between tasks and environments requiring a spatial or visual representation. The greater the number of criteria is an indicator that a visual depictive representation is more likely required rather than a quantitative spatial representation. Although they propose that these representations fall on a continuous spectrum, we ignore that for now as our theory states that the agent must make a commitment to one or another representation. The criteria are (with our slight modification):

(1) Number of different types of spatial relationships (direction, distance, orientation, topology, size)
(2) Number of spatial relationships
(3) Specificity of the shape
(4) Specificity of the color

The following are the specific spatial and visual subtasks, one or more of which, support a general task. We provide a figure to illustrate some of these subtasks. Before looking at the figure, first try "imagining" the situation.

*(a)* *Infer **global** spatial relationships (i.e. direction, distance, orientation, topology, size) or visual properties (i.e. new shapes or color) derived from the combination of objects and their perceived or retrieved **local** spatial relationships*.

Example 1 (Figure 5-1a, NOTE: "you" are the "X"): Target A is 500m to your left front. Target B is 250 meters to your right. What is the direction between Target A and Target B? What is the distance between Target A and Target B? In this example, there are two different types of spatial relationships (direction and distance) and three spatial relationships (X-A, X-B, A-B). Shape and color are not required so a spatial representation is sufficient for this example.

Example 2 (Figure 5-1b): What is the angle between a ray from you to target A and a ray between you and target B? In addition to the direction and distance relationships, you now have an orientation to consider. This task is again spatial, but with an increase in the type and number of spatial relationships moves it closer to a task where visual imagery may be necessary.

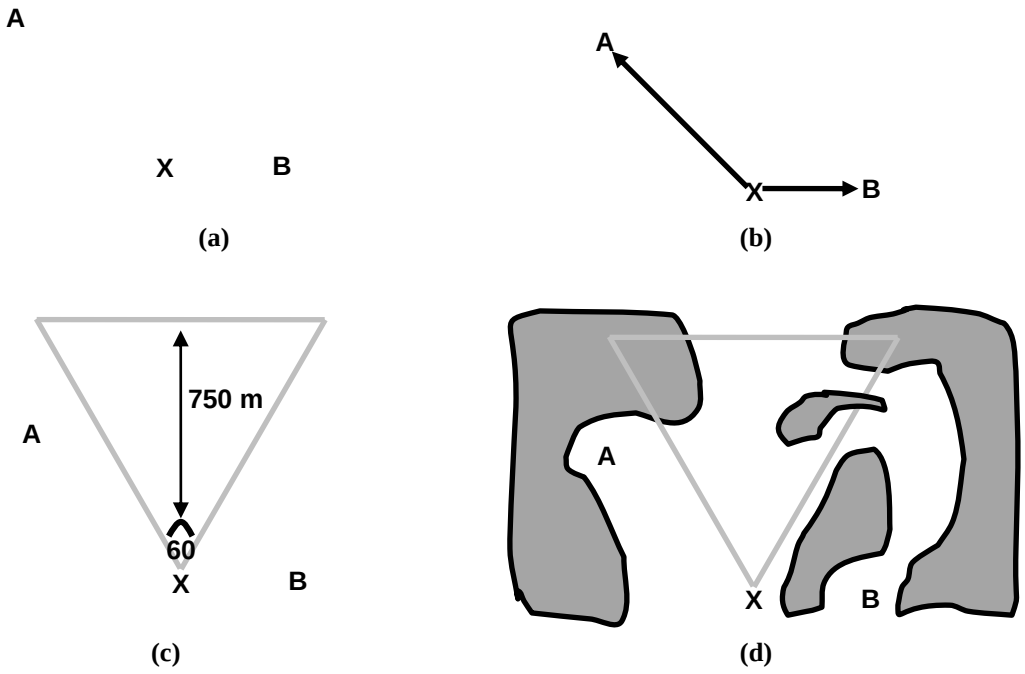



**Figure 5-1: Spatial and Visual Task Spectrum**

Example 3 (Figure 5-1c): Add a triangle with a vertex at your location and a base to your direct front. The interior angle of the vertex is 60 degrees and the height of the triangle is 750 meters. Is target A inside the triangle? Target B? Again, we have increased both the type of spatial relationships (triangle size and topology—inside/outside) and the number of spatial relationships.

Example 4 (Figure 5-1d): Superimpose what you currently have imagined onto the background in Figure 5-1d. Ignoring the triangle, are A and B in the same topological space (i.e. can you get from A to B without crossing a gray area)? The background contains non-convex shapes and more spatial relationships to consider. Visual imagery and a depictive representation are likely to be necessary for this task.

Example 5: Are there any parts of a head of Iceberg lettuce that are a darker green than any parts of a Christmas tree? [Answer: yes – if the tree has some light green ornaments on it.] In this example, you have to combine the two objects in a visual depictive representation to make the specific color comparison.

**(b)** *Infer results after a transformation*.

Example 6 (Figure 5-1c Notice we are back to figure "c"): Rotate the triangle counterclockwise 45 degrees. Is target A inside it? Now rotate the triangle clockwise 90 degrees. Is target B inside it? Without specific shape, the spatial representation is sufficient.

Example 7 (Figure 5-1d): Starting from the triangle's original orientation (i.e. north), rotate the triangle clockwise 90 degrees. Is there an enclosed gray region between you and target B? Similar to example 4, the task might benefit

significantly from visual imagery because of the specific shape and multiple types and number of spatial relationships.

***(c)*** *Retrieve the spatial or visual properties from an exemplar.*

Example 8 (Figure 5-1d): Imagine a path from A to B avoiding all enclosed regions. Is there a location where the path turns approximately 90 degrees? Retrieving spatial or visual properties of an exemplar requires visual imagery as the exemplar's specific shape and, possibly color, are involved. In this example, the imagined path is a specific exemplar.

Example 9: In what hand does the Statue of Liberty hold the torch? A specific instance of a visual object (i.e. Statue of Liberty) likely necessitates a visual depictive representation.

***(d)*** *Retrieve a prototypical object's spatial or visual properties when the property was not explicitly encoded*

Example 10: Does the letter 'A' have an enclosed space? The letter 'B'? 'X'? In these examples, the prototypical letter may have an explicit symbolic description (e.g. the letter 'X' is two intersecting lines), but not enough information to perform the reasoning in the question. Since specific shape is involved, a visual depictive representation is likely necessary for the task.

In summary, the previous discussion highlights the general tasks where imagery is useful for reasoning to include filling in missing spatial and visual details of a situation, recognizing novel visual features and spatial properties, analyzing or rehearsing the outcome of an action before executing it, and replaying a previous event to inform a future decision. One or more subtasks support these tasks and may require spatial or visual imagery depending on the number and types of spatial relationships and the specificity of the shape and color. The agent decides what representation is suitable for

the task. Subtasks include, but are not limited to, inferring global spatial relationships or visual features from perceived or retrieved local spatial relationships, inferring new spatial and visual properties after a transform, and retrieving a spatial or visual property from an exemplar or a prototypical object where the feature was not explicitly encoded. The environment may or may not play a role in determining the characteristics of the tasks. We now summarize our three experimental domains as concrete examples. We will refer to these domains when we discuss the architecture in the next chapter and revisit the tasks in more detail when we evaluate the architecture in Chapter 7.

## 5.2 Geometry Gymnastics

The geometry problem derives from Larkin's and Simon's (1987) work demonstrating the computational advantage of diagrams. In one of the problems they investigate, the agent must locate visual properties (e.g. vertices, line segments, triangles) and infer relationships (e.g. angles) that initial task knowledge does not specify. The problem, shown in Figure 5-2, consists of four lines (A, B, C, D). Line A is parallel to line B and line C intersects line A. Line D bisects the line segment formed by the intersection of line C with lines A and B. The goal is to show that the two triangles formed are congruent. To prove congruency, the model must employ a basic geometry rule, such as the angle-side-angle (ASA) rule. The ASA rule states if two angles and the included side of a triangle are congruent to two angles and the included side of another triangle, then the two triangles are congruent. In Figure 5-2, the model must show E1=E2, e1=e2, and c=b.

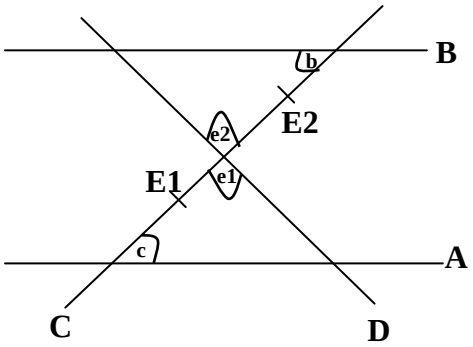



**Figure 5-2: Geometry Problem**

The environment is irrelevant in this task, as it only requires internal problem solving. The general imagery tasks are to fill in the spatial and visual details and recognize novel features and spatial relationships by combining objects (i.e. lines) based on their local spatial (i.e. direction, distance, orientation) relationships. Another way to look at the problem statement is that line B is in front of line A. Line C is in between line A and line B and oriented counterclockwise some random orientation between 30 and 60 degrees from line A. Line D is also in between line A and line B oriented clockwise from line A. Because the number and type of spatial relationships are more than a few but specific shape and color are not required, this task only requires spatial imagery. In the results chapter we will discuss the comparison between an exclusive symbolic and a combined symbolic/quantitative spatial implementation.

## 5.3 Alphabet Soup

An experiment from Thompson et al. (in press) motivates the second domain. In their experiment, the subject hears a letter from the English alphabet, and the investigator asks the subject to visualize the letter in its uppercase format (Figure 5-3). Next, the subject hears a cue, such as "curve," "enclosed-space," or "symmetry" and indicates (by pushing a button) whether the letter has the particular feature. For example, the letter 'A' has an enclosed space and vertical symmetry while 'U' has a curve and vertical symmetry. The Soar agent also "hears" a question, visualizes the letter, searches for the desired feature, and then "verbally" responds.

While there are environmental cues in this domain (i.e. the agent "hears" a question), like the geometry problem, it is primarily an internal problem-solving task. The general imagery task is to fill in the visual details by retrieving a specific feature from a prototypical, uppercase letter. As specific shape is necessary to infer the visual features, this task focuses on the depictive representation. Unlike the geometry domain, symbolic or quantitative representations may have significant challenges, both computationally and functionally, solving this task without explicitly encoding every feature.


**(a)** **(b)**

**Figure 5-3: Alphabet Experiment**

## 5.4 Scouts Out

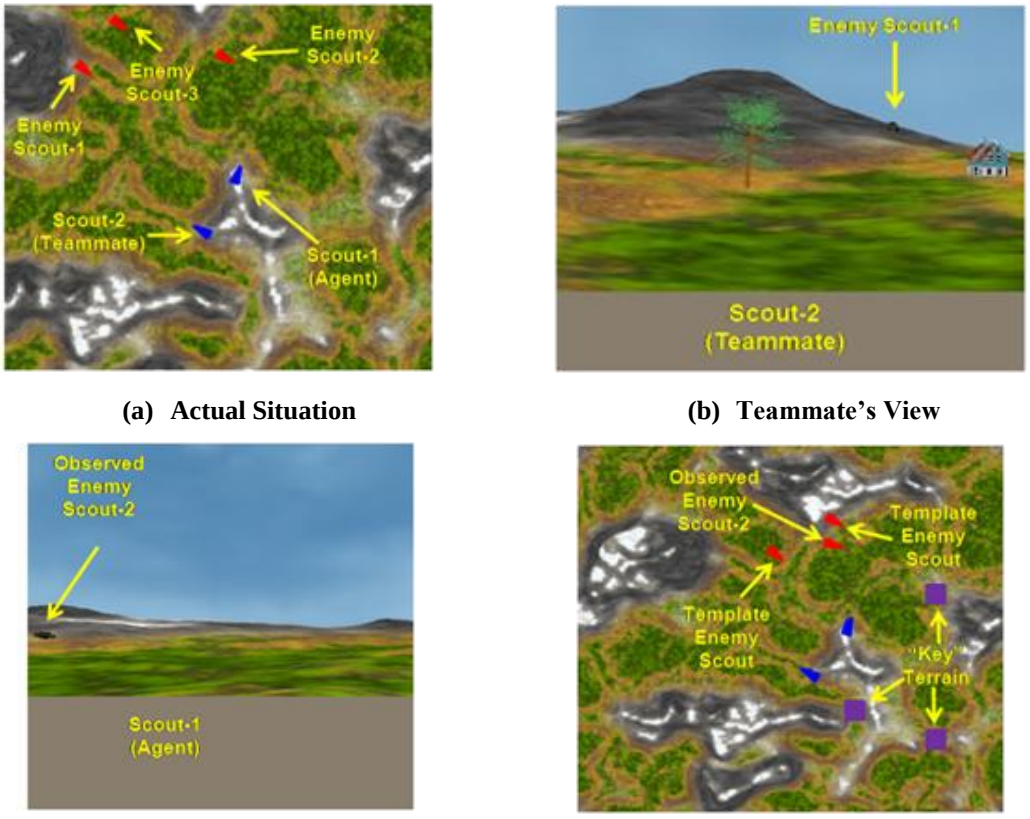
This evaluation environment is motivated by the U.S. Army's work in developing robotic scouts to provide situational awareness for a mixed manned/unmanned military force (Jaczkowski, 2002). Supporting intelligent tactical behavior, rather than serving as a sensor platform on wheels, is one of the goals for the robotic scouts. That is, in addition to autonomously maneuvering to a position and transmitting video data, we would like the scouts to coordinate and attempt to improve their positions based on sound tactical behavior.

In support of this effort, we built a simulation to model a section of two scout vehicles that must cooperate to maintain visual contact with an approaching enemy's three-vehicle reconnaissance element (Figure 5-4a). One scout, the section lead, is a Soar agent. The other scout, the teammate, is scripted. The team's primary goal is to keep its commander informed of the opposing force's movements by periodically sending observation reports (through the lead) containing their best assessment of the enemy's location. The agent cannot observe its teammate because of terrain occlusions. However, the teammate periodically sends messages regarding its position. The teammate scans the area in front of it and sends reports to its lead when it observes enemy vehicles (Figure 5-4b). The teammate also responds to orders from the lead to reorient its view. The agent can look at the environment or its map (Figure 5-4c-d) and can reorient its view. We assume the agent and its teammate can distinguish enemy vehicles from other objects. However, the agent has to decide whether a sighted or reported enemy is a new or previously identified entity.

To motivate the capabilities of multiple-representations, consider how the agent makes decisions in this domain. Typically, a scout leader follows these steps after initial visual contact (Army, 2002).

(1) Deploy and report
(2) Analyze the situation
(3) Choose and execute a course of action

Analyzing the situation involves reasoning about known friendly and enemy locations and orientations, terrain, and obstacles. If the scout lead does not know the locations of all expected enemy, then he might hypothesize where other enemy vehicles are and imagine their positions (Figure 5-4d). Based on the analysis, the scout leader then decides if he should reorient himself, his teammate, or both.




**(a) Actual Situation** **(b) Teammate's View**

**(c) Agent's View** **(d) Agent's Perceived Map / Imagined Situation**

**Figure 5-4: Scout Domain**

Analysis often involves visualizing the situation and mentally simulating alternatives. Military leaders rely on imagery to assist with decisions in the "fog of war." The U.S. Army's doctrine even goes so far to state:

> Visualize means to create and think in mental images. Human beings do not normally think in terms of data, or even knowledge; they generally think in terms of ideas or images—mental pictures of a given situation (Army, 2003).

Using spatial and visual imagery, an agent can imagine each observed entity's map icon on its external map. If the agent is confident in the information, it can write it on the external map, in effect making it persist. As information changes the agent updates the map, keeping its perceived map of the situation up to date. Note that the agent may, but does not have to, keep the location and orientation in its head. It can simply "look" at its external map and "read" the information. In this sense, then the map serves as an external store. Using the external map as perceptual background, the agent can then imagine key terrain (enemy objectives), hypothesized enemy, possible enemy paths, its viewpoint, and its teammate's viewpoint. It can then imagine alternative course of action by simulating different viewpoints.

This domain has environmentally rich spatial (e.g. relationships between entities, obstacles, terrain, etc.) and visual (e.g. terrain's topological shape and color) properties. In addition to the general imagery tasks of filling in the spatial and visual details and recognizing novel shapes by combining perceptual input with retrieved memories, imagery is used to analyze the outcome of an action before executing the action (e.g. imagining different viewpoints for the teammate and self). The sub-tasks cover both spatial and visual tasks with similarities to the examples provided in the beginning of this chapter.

In summary, decision-making proceeds by combining perceptual representations with task specific and declarative knowledge to construct an imagined scene. Analysis emerges through the manipulation of symbolic, quantitative spatial, and visual depictive representations. Retrieval or inspection of the resulting representations then provides new information that the agent uses to reason with to produce action in the environment.

# Chapter 6

# Architectural Design

The previous chapters present the background necessary to appreciate the architectural design decisions. Although we introduce the theory and design space constraints at once, our research strategy is an iterative process. First, we analyze specific behavioral phenomena supporting a desired functionality. In our case, it was how humans use mental imagery, or visualization techniques, to make decisions and solve problems. We then determine plausible computational approaches motivated by the behavioral and biological evidence. Next, we design and implement a complete (i.e. perception, cognition, action), although rudimentary software system. Finally, we evaluate the system starting with simple tasks and progress to more complex scenarios. The evaluation process drives our future direction and requirements for subsequent iterations.

For example, we initially did not consider how perceptual mechanisms constrained imagery. As we investigated the literature, however, it became clear that imagery and visual processing are not disjoint components, but rather use and share similar structures and processes. As part of that oversight, we did not include the visual depictive representation, as it initially seemed odd to us that humans would resort to such a low-level representation for reasoning after perception performed so much work extracting abstract representations. It became evident, especially when we began evaluating the requirements for the alphabet experiment, that a depictive representation was not only useful, but appeared necessary to achieve the desired functionality. Finally,

when we evaluated the system in a more perceptually demanding environment (i.e. the Scout domain), where bottom-up, visual processing and imagery must cooperate and share resources, we had to look at issues such as the differences between perceived and imagined objects, synchronization of processes, race conditions between perception and imagery, and truth maintenance issues. While we do not claim to have implemented all functionality that humans show during spatial and visual imagery, the goal has been to design and implement a *complete* and *general* architectural framework with a few demonstrated capabilities that can motivate future work.

Soar and the Spatial-Visual Imagery (SVI) module are the two major components in the architecture (Figure 6-1). Soar encompasses the symbolic representation and computations. SVI includes the quantitative spatial and visual depictive representations and processes. It encapsulates high-level visual perception, spatial, and visual imagery. Our modeling of visual perception, to include the separation between "what" and "where" pathways, is theoretical and an approximation, but we include it for completeness. The architecture makes a distinction between memories (rectangles) and processes (rounded rectangles), and the terminology is either Kosslyn's et al. (2006) or our own.

We present the architecture as follows. First, we provide an overview focusing primarily on the memories and corresponding data structures associated with Soar and SVI (Soar+SVI). Next, we suggest, primarily from a theoretical perspective, how perceptual visual processing emerges. We then describe our implementation of spatial and visual imagery processing. For more details, Appendix B illustrates some algorithms for manipulating and inspecting the visual depictive representation. Appendix C details the software engineering aspects and provides examples of the Soar symbolic structures used to represent and control imagery processing.

## 6.1 Soar

Soar (Laird, 2008; Lehman, Laird, & Rosenbloom, 2006) provides a fixed set of symbolic memories and processes (top of Figure 6-1). The symbolic memories include a declarative, short-term memory (STM), a procedural long-term memory (LTM), two long-term, declarative memories (episodic and semantic, not shown in Figure 6-1), and

two learning mechanisms (chunking and reinforcement learning). The short-term memory is a graph structure representing the agent's knowledge of its goals and current state. A symbolic structure in Soar may represent many things, including concepts (e.g., cheetahs run fast) or objects in the world. Within the Soar+SVI architecture, there are special annotated symbols that represent an object and its explicit spatial and visual properties. We call these symbols *visual symbols*. These symbols arise from perception, activation of a previously stored memory, or results from an imagery inspection. Visual symbols may be associated with other, non-visual symbols. We will discuss visual symbols in more detail later.

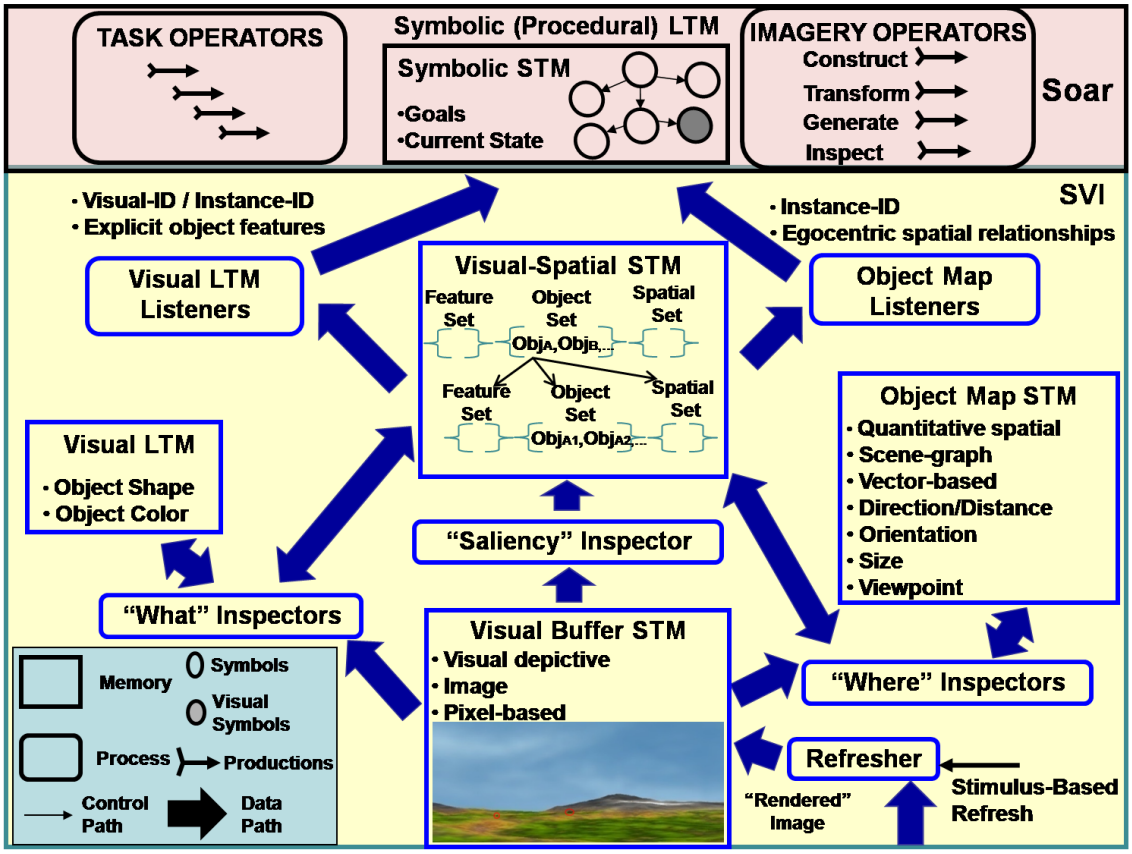



**Figure 6-1: Architecture Overview**

Soar's procedural long-term memory is a set of productions that control behavior. Each production has a set of left-hand side (LHS) conditions and right-hand side (RHS) actions. If a symbolic pattern in STM matches with the LHS of a production, then the production "fires," creating or removing symbolic structures in STM based on the RHS

actions of the production. During each phase of Soar's processing cycle (described next), all matching productions fire in parallel.

Soar's processing cycle is based on the problem space computational model (PSCM) organizing an agent's knowledge into a set of *states*, and a set of *operators*, instantiations of which move the agent to different states. An agent, by iteratively selecting and applying operators, effectively conducts a search through its problem space. The processing or decision cycle (Figure 6-2) is hypothesized to be approximately 50 milliseconds in humans, but is much faster in the actual implementation.

The decision cycle begins with an input phase where an agent's current perceptions augment a fixed structure in STM called the *input-link*. The elaboration phase provides an opportunity, through the matching and firing of productions or retrieval from a declarative LTM, to elaborate all knowledge relevant to the current situation, propose potential operators, and create preferences for those operators. After the elaboration phase reaches *quiescence*, the operator selection, or decision phase, selects an operator from the set of proposed operators. The selection is based on the operators' preferences. If knowledge is inadequate to choose between the different operators, an *impasse* occurs and the architecture creates a subgoal enabling further reasoning.

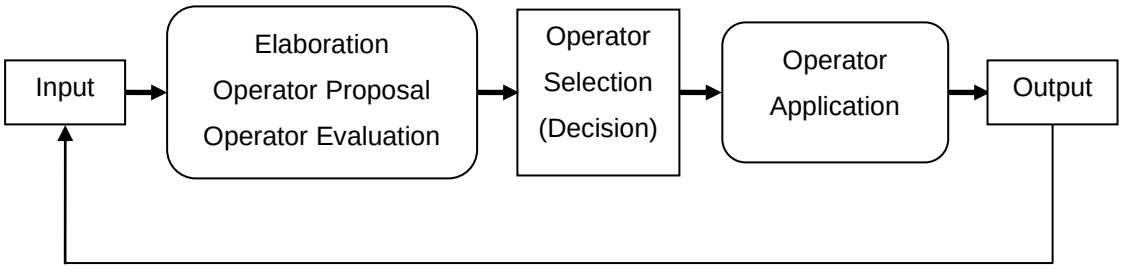



**Figure 6-2: The Soar Decision Cycle**

After the decision phase, if there is a selected operator it applies, again through the firing of one or more productions, making persistent changes to short-term memory. These changes either update the agent's internal model or create motor commands on a fixed structure in short-term memory called the *output-link*. During the output phase, a process external to Soar reads the commands placed on Soar's output-link and produces action in the environment. In the Soar+SVI system, the SVI module filters all output-link commands. If the command is an imagery action then it is transmitted to the appropriate

SVI component handling the command. Otherwise, SVI passes it to the appropriate motor module.

## 6.2 SVI

SVI encapsulates the fixed memories and processes to support high-level visual perception, spatial imagery, and visual imagery (bottom of Figure 6-1). The memories include short-term memories for the quantitative spatial (*Object Map)* and visual depictive representations (*Visual Buffer)*, a long-term memory (*Visual LTM)* that stores the shape and color of prototypical objects or specific instances that the agent has previously seen in the environment, and a short-term memory (*Visual-Spatial STM)* that binds the "what" and "where" pathways.

### 6.2.1 Memories

#### 6.2.1.1 Visual Buffer

The *Visual Buffer* (bottom of Figure 6-1) is a depictive, short-term memory activated from either bottom-up, visual-perception or top-down imagery processing. Space is inherent in the structure of the depictive representation and the encoding is of information rather than a denotation of information. The depiction as a whole represents shape, size, orientation, location, and color.

Computationally, the Visual Buffer is a set of 2D image data structures where a single image, ***I***, has a set of picture elements, or *pixels*, (**x,y; *I*(x,y)**). The first parameter, **(x,y)**, is the pixel location and the second parameter, ***I*(x,y)**, is the pixel value of ***I*** at location **(x,y).** There is always at least one image in the set, the base image or Visual Buffer layer (*vb-layer*) zero, representing the perceived scene from an egocentric viewpoint or an imagined scene from an imagined viewpoint. Visual perception or imagery may create additional, ephemeral images in the set that extract a subset of the base image (e.g. edges, marked regions). These subsequent image layers serve as an attention mechanism to support further computations. Algorithms, encoded in the "*What*" or "*Where*" inspectors (bottom of Figure 6-1) or the *VBManipulator* (shown later) process each image separately. The algorithms used to process the image(s) are algebraic

(e.g. edge detectors, filter masks, rotation, scaling), or ones that take advantage of the topological structure using pixel-level rewrites (Furnas, 1990, 1991; Furnas et al., 2000).

For example, in the alphabet experiment, a Hough transform is used to detect curves on letters and pixel-level rewrites are used to infer enclosed spaces. Both techniques require the instantiation of additional image layers to support reasoning. In the scout domain, the system creates a separate image for each enemy and "key terrain" toward which the agent hypothesizes the particular enemy is maneuvering. Figure 6-3 shows the agent's combined perceived and imagined scene inside the Visual Buffer box. The middle image represents the agent's imagined representation of one enemy maneuvering toward a piece of terrain along with a distance field flood and the hypothesized path (shown in orange). The image serves as an attention mechanism in that the agent is focused on a particular enemy/key-terrain pair and the path between them. The third image in the figure's upper right corner represents the portion of that path the agent hypothesizes it can view based on its current location orientation, and imagined field of view.

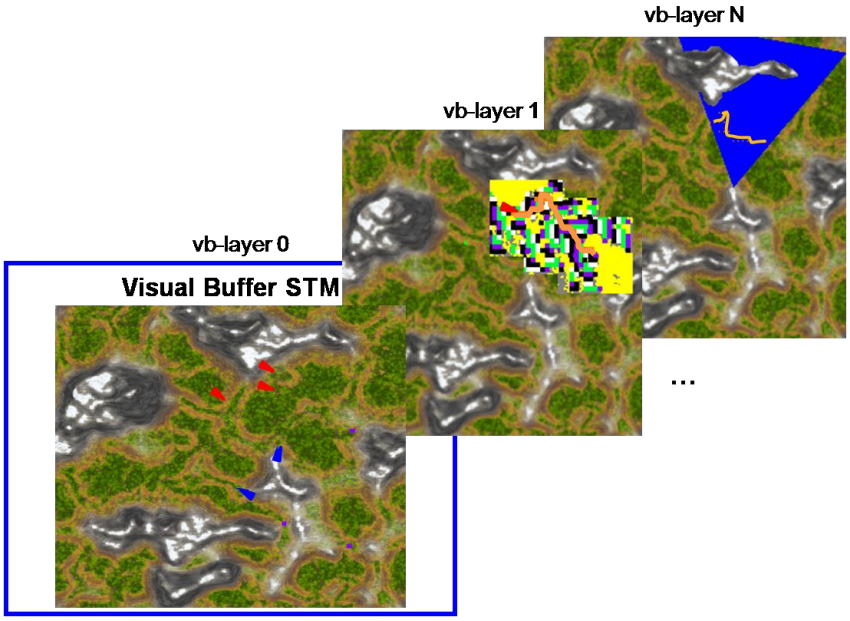



**Figure 6-3: Visual Buffer is a "Set" of Images**

### 6.2.1.2 Object Map

The *Object Map* (middle right of Figure 6-1) is a short-term memory that maintains the quantitative spatial representation of the objects in the currently perceived

or imagined scene. This representation fixes an object's location (direction and distance from a fixed frame of reference), orientation, and size in space and includes a viewpoint to facilitate the generation of a depictive image. The computational processes that infer knowledge from this representation are sentential, mathematical equations. It also serves as an intermediate format when moving from a symbolic to a depictive representation. The Object Map's structure is sufficient for reasoning about objects' relative direction, distance, orientation, size, and general topology. However, if reasoning of spatial properties involves a non-convex shape or the reasoning is about an object's visual features, a depictive representation in the Visual Buffer is required.

A key design decision was determining the Object Map's data structure. The structure had to support the quantitative spatial representation for spatial reasoning tasks and the rendering of a rasterized image for the Visual Buffer's depictive representation. For spatial reasoning, we desired a structure supporting hierarchical, containment relationships (e.g., a line is part of a geometric figure, a gun is part of a tank) and spatial relationships (e.g. line-C is in between line-A and line-B, enemy A is 500 meters to my left front). Together, the hierarchical containment and spatial relationship properties support spatial reasoning between objects and an object's parts (e.g. what is the relationship between the front wheel of a car and the steering wheel? Between the steering wheel and the stop sign?). Rendering to a rasterized image requires the ability to combine specific shape (i.e. vertices and indices) and color, spatial relationships, and a privileged viewpoint so that the image can be generated from a specific perspective.

We chose to implement the Object Map with a scene-graph data structure and a viewpoint, or camera (Eberly, 2005). Every node in the scene-graph is an object or group of objects representing a portion of convex space in the perceived or imagined scene with the root node representing the entire space (Figure 6-4). The leaf nodes of the graph contain an object's shape (i.e. three-dimensional mesh of vertices and indices) and color (i.e. a red, green, blue vector) to support rendering to an image. Intermediate nodes represent the composition of one or more objects. The structure is called a graph (rather than a tree) because multiple leaf nodes may share the vertices and color.

As an SVI convention, the root node of the Object Map's scene graph represents the current scene (Figure 6-4). The root's first child contains the agent's spatial

information (location, orientation), the second child includes any perceived background information, such as terrain, and the third child contains all the salient or visual objects in the agent's scene. Figure 6-4 shows the number of visual objects to be N. From a psychological perspective, N is hypothesized to be four to five objects based on working memory capacity (Jonides et al., 2008; Pylyshyn, 2001). Note that there may be more objects in the environment, but N simply represents those objects that perception has determined "salient" or imagery has added. Each visual object may be a single entity (i.e. a tank) or several entities (i.e. a group of tanks).

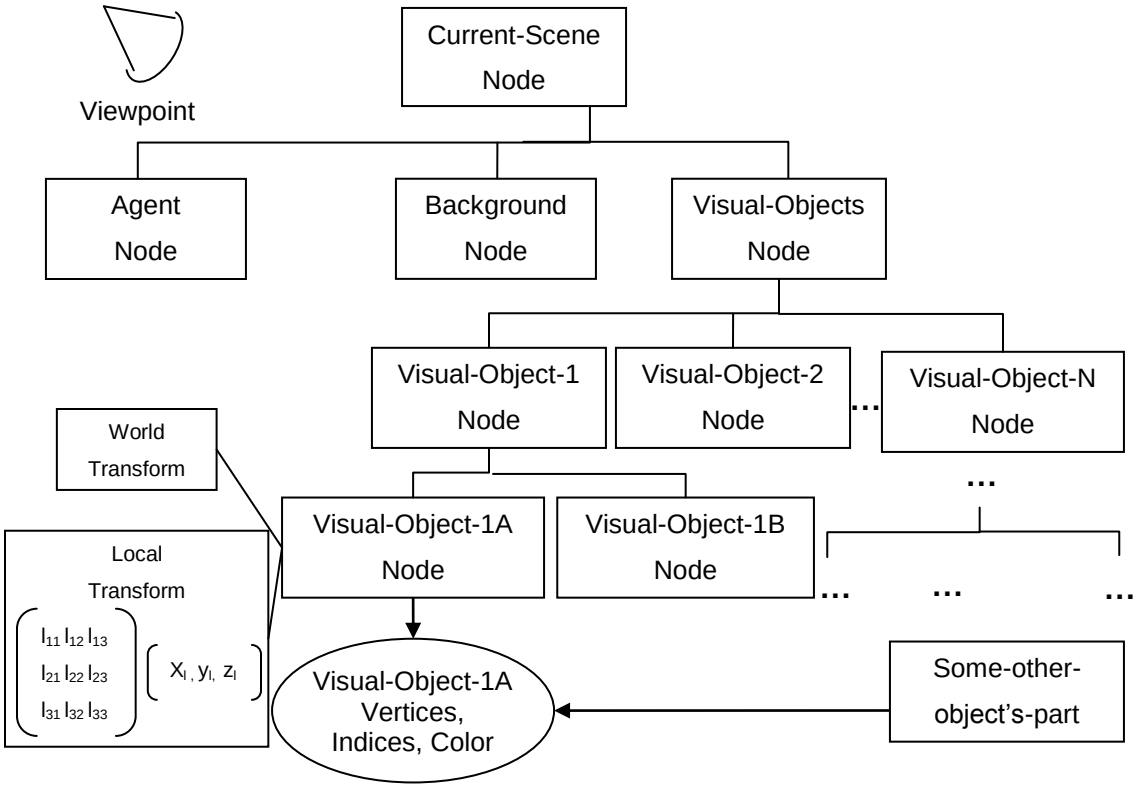



**Figure 6-4: Scene-Graph Data Structure**

SVI distinguishes between *primitive* and *composite* objects. Primitive objects, such as Visual-Object-1A's Node, have a single child encoding the vertices and color. Composite objects are composed of one or more primitive objects. The general shape, or "bounding volume," of a primitive object is computed from its vertices. The general shape may be a convex hull, an oriented-bounding box, or a sphere. Each intermediate

scene-graph node, or composite object, captures the general, convex shape of an object it represents based on the combined general shape of its child objects.

Every node in the scene graph encapsulates a *local* and *world* transformation where a transformation includes a 1x3 translation vector and a 3x3 rotation and scale matrix. A local transformation represents the direction, distance, orientation, and size of an object relative to its parent object. A world transformation is relative to some fixed, global frame of reference. For example, in Figure 6-4, Visual-Object-1's Node is composed of two parts, Visual-Object-1A and Visual-Object-1B. Visual-Object-1A's and Visual-Object-1B's local transformation represent the spatial relationship (i.e. direction, distance, orientation, size) between them relative to Visual-Object-1. The two parts' world transformation represents their global location, orientation, and size computed from Visual-Object-1's world transformation. That is,

$$\text{World}_{\text{Visual-Object-1A}} = \text{World}_{\text{Visual-Object-1}} * \text{Local}_{\text{Visual-Object-1A}}$$

The general shape and transformations are computed recursively at run-time by traversing through the graph. Note that the "world" transformation does not necessarily have to be "global" coordinates. Rather, an alternative is to have the agent serve as the "world" origin and the transformations computed relative to this origin. Of course, in practice to compute an actual world location one would have to know the agent's location, perhaps through localization techniques or a global positioning system.

#### 6.2.1.3 Visual Long-Term Memory

The remaining memories in SVI are not associated with a particular representation but are indirectly involved in reasoning. *Visual long-term memory* (VLTM) contains prototypical objects and specific instances encoded from previous experiences. VLTM is implemented as a hash table (Figure 6-5). A symbolic, *visual-id*, indexes each object. Each entry in the table is an object's scene-graph representation. The scene-graph in VLTM is distinct from the scene-graph in the Object Map as it is not an instance in the current perceived or imagined scene but rather a memory of a prototypical object's shape and color. Another distinguishing characteristic is that unlike an object instance in the Object Map, a VLTM object does not have a fixed frame of reference in

space relative to the agent (egocentric) or another object (allocentric). Rather the space representation in VLTM serves only to specify an object's configuration properties. To become an instance in the current scene, imagery activates this VLTM representation.

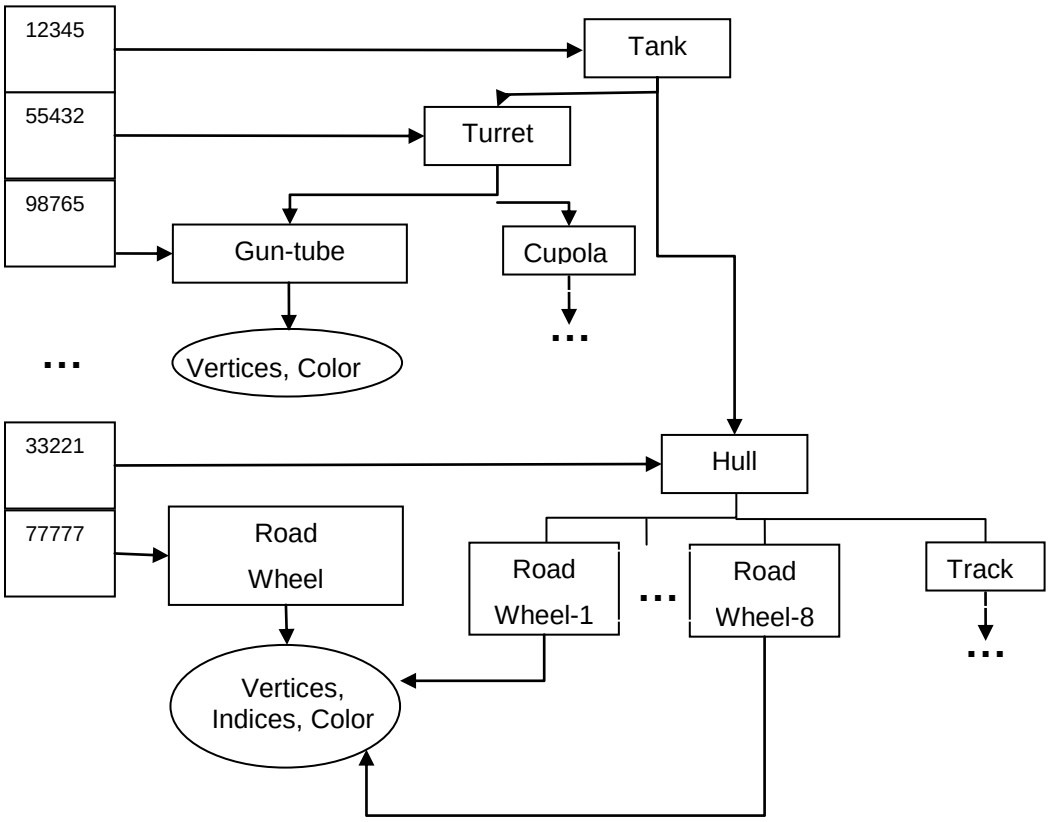



**Figure 6-5: Visual Long-Term Memory**

Similar to the Object Map, VLTM distinguishes between primitive and composite objects. For example, VLTM stores a tank from the Scout domain as shown in Figure 6-5. The tank has a hull and a turret, the hull has road wheels and tracks. The advantage of this representation is its compactness. Only the leaf-node entries store the vertices and color of the primitive objects (i.e. road wheel). All other node entries are simply pointers and transformations describing the composition and relationships between the object's parts. Together the structure determines the object's shape. Details in the representation that are missing are just missing. Someone not familiar with a tank may think of it as a cylinder on top of a rectangle rather than containing all the features that a tank expert has. Thus, the primitive parts simply have fewer vertices and color details that the agent may elaborate as it experiences more objects in the world of a similar type.

The VLTM structure enables an agent to imagine an object (i.e. tank) or an object's parts (i.e. a tank's gun tube or road wheel) by simply traversing the nodes indexed from the visual-id and instantiating (i.e. copying) each child node to the Object Map. The vertices and color are shared between the instantiated objects in the Object Map, and the objects in VLTM. The vertices are involved in computations when determining the object's general shape for spatial imagery (e.g. computing the convex hull) or generating the depictive image in the Visual Buffer for visual imagery.

#### 6.2.1.4 Visual-Spatial Short-term Memory

*Visual-Spatial short-term memory* (VS-STM) is a shared memory that effectively binds the "what" and "where" pathways and serves as a temporary symbolic store between Soar and SVI (center of Figure 6-1). It is a hierarchical structure with the top-level representing the sets of salient, visual objects, spatial properties, and visual features that apply to the current scene--either perceived or imagined. Each salient object in the visual object set may have subsequent levels in the hierarchy with its own feature, object, and spatial sets (Figure 6-6). Visual processing expands the top-level sets and subsequent layers as details in the scene are elaborated. During imagery, results of processing also expand these sets in response to specific operations. Each visual-object in VS-STM has a corresponding *instance-id* distinguishing it as a salient object in the scene. If the visual-object is recognized it will also have a *visual-id*. Note that the Visual Object in VS-STM is a separate structure from the Visual-Object *Node* in the Object Map's scene graph (Figure 6-4). A VS-STM Visual Object has direct access to its corresponding scene graph node but contains information not represented in this node such as any visual or spatial properties elaborated during perception or imagery, whether it has been recognized (i.e. visual-id), and a *marking-color* and *vb-layer*. Our description of visual perception (discussed next) will discuss the marking-color and vb-layer. This VS-STM Visual Object then binds the "what" and "where" pathways.

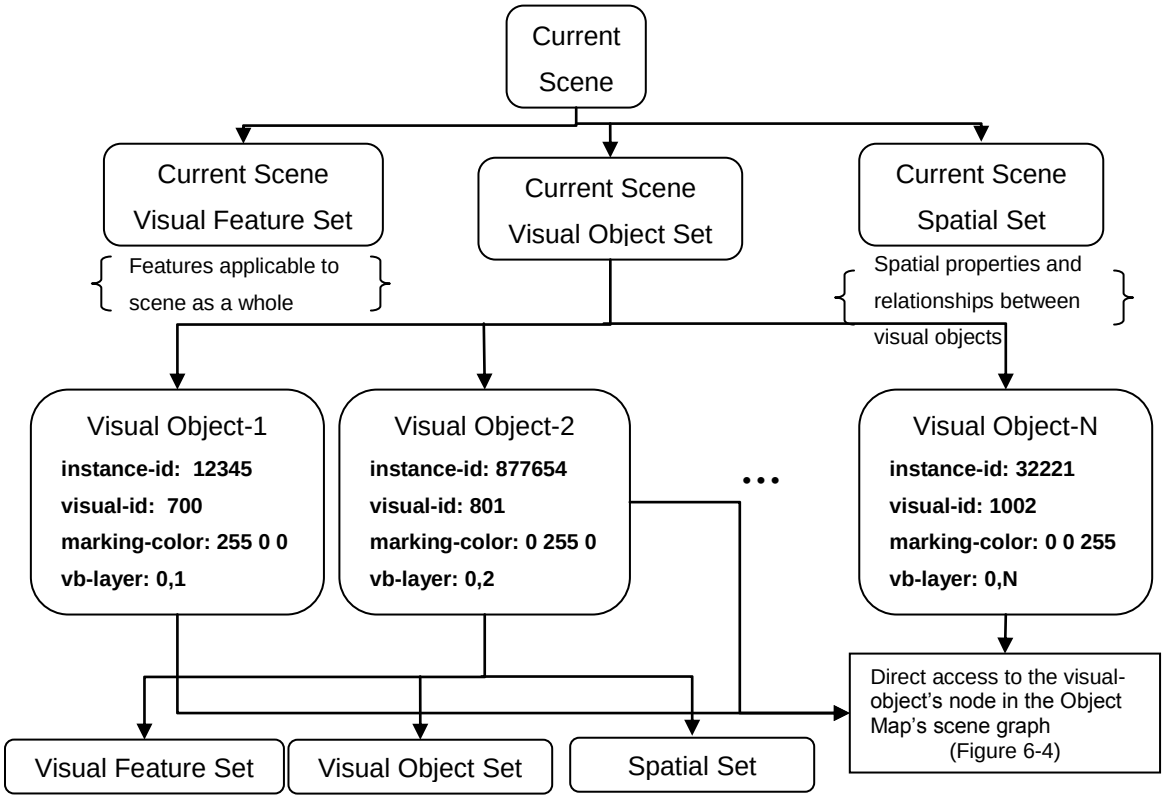



**Figure 6-6: Visual-Spatial Short-Term Memory**

## 6.2.2 Processes

### 6.2.2.1 Visual Perception

Again, our theory of visual perception is theoretical, but we include it to emphasize imagery's integration with visual mechanisms. Our implementation does model the flow of information as described here. A *Refresher* process activates the Visual Buffer from retinal stimulus (bottom right of Figure 6-1). Upon activation, a two-pass operation is performed on the depiction. During the first step, a *Saliency Inspector* determines and marks the salient regions, or objects, in the current scene by creating a separate image, or *vb-layer*, for each object. Each image is "colored" with a unique marking where the colored region corresponds to the salient object's contour and interior in the perceived image (Ullman, 1996). The Saliency Inspector creates a *Visual Object* structure in VS-STM for each salient object and augments it with a unique *instance-id*, *marking-color*, and associated vb-layer (Figure 6-6). The instance-id is similar to Pylyshyn's (2001) concept of a visual index.

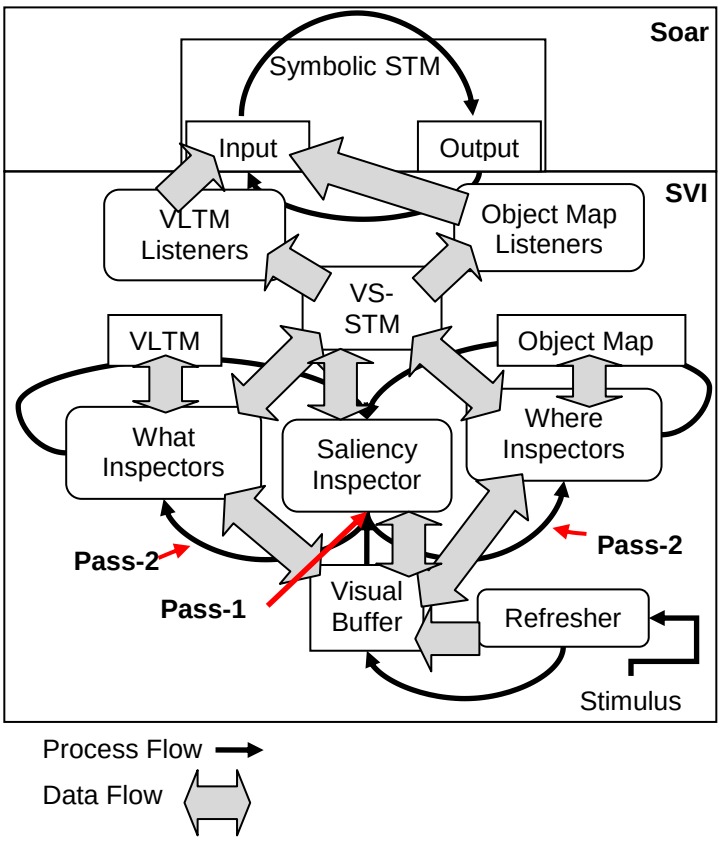



**Figure 6-7: "Bottom-up" Visual Processing and Data Flow**

During the second pass, two parallel processes initiate a more detailed inspection of the depictive representation, focusing their attention on the marked objects in the images in the Visual Buffer layers (Figure 6-7). The *What Inspectors* are responsible for extracting object features to support recognition by matching the features with a shape and color representation in VLTM. If the object is recognized, its associated visual-id from VLTM is stored in VS-STM (Figure 6-6).

Simultaneously, the *Where Inspectors* extract the egocentric direction and distance (i.e. location), orientation, and size of the objects from the Visual Buffer, build the Object Map's scene-graph, and update the VS-STM's current scene spatial set (Figure 6-6). Initially, the "Where Inspectors" encode the general shape of the object in the scene graph as a "blob" based on the marked region in the Visual Buffer. If the visual object has been recognized by the "What Inspectors" (i.e. it has a visual-id in VS-STM), the "Where Inspectors" associate the leaf nodes of the Object Map with their corresponding vertices in VLTM.

A set of listeners along the "what" and "where" pathways (VLTM and Object Map Listeners respectively) monitor updates to VS-STM and consolidate the results for input to Soar (Figure 6-1 & Figure 6-7). Operators in Soar's procedural memory, executing in a separate control path, attend to the listeners' input and associate it with existing knowledge to identify the object. For example, a recognized tank object may be associated with the fact that it is an enemy vehicle. If the visual object arriving on Soar's input-link does not have an associated visual-id (i.e. it is not recognized), imagery processing may commence in Soar to assist in recognition of the object. Soar encodes the visual objects in the perceived scene along with their egocentric direction, distance, and orientation. Note that these "automatically," extracted spatial relationships are relative to the agent.[8] They do not include the relationships between every pair of salient objects in the scene. This type of inference requires imagery.

In practice, the simulation environment provides the Saliency Inspector a list of scene graph nodes representing the "salient" objects that the agent can currently observe. The simulation makes the determination of what the agent can and cannot observe. The first scene graph node in this list is always the "background" node, which may represent terrain and other "non-salient" objects (i.e. trees, buildings), etc. The Saliency Inspector instantiates the Visual Object structure in VS-STM and provides the "What" and "Where" Inspectors the list of scene graph nodes. The "What Inspectors" recognition process is a simple "string match" and the "Where Inspectors" build the internal Object Map structure from the provided list of scene graph nodes. For debugging purposes, the Refresher renders the internal scene graph from the agent's current viewpoint. The remaining information flow proceeds as previously discussed. Although the implementation of bottom-up visual processing is ad-hoc for now, it forces us to consider how visual perception and imagery interact.

#### 6.2.2.2 Spatial and Visual Imagery

Given the above discussion, we can now describe how the architecture supports spatial and visual imagery processing. We will provide an overview of the processing, discuss

[8]There may be some non-egocentric relationships automatically extracted during bottom-up perception such as several objects falling in a line.

the spatial and visual knowledge representation in Soar, and then elaborate on how SVI processes each imagery operator (*construct, transform, generate,* and *inspect*). We will use examples from both our experimental domains and the following "place-setting" task:

> *A Soar agent is setting the table for dinner. Its current goal is to set one place setting. In order to accomplish the goal it has to set each individual object (napkin, fork, plate, etc). It prefers to set the center object (i.e. plate) first so it can place the other objects relative to the center.*

An agent uses imagery when, in the context of its current goal, a spatial relationship or visual feature is not directly accessible from symbolic knowledge. For example, in the "place-setting" task the agent is trying to determine the center object; in the geometry problem, the agent is trying to find vertices, line segments, angles, and triangles; in the alphabet experiment, the agent is trying to determine if the letter has a specific feature; and in the Scout domain, the agent is trying to determine where to orient its team for adequate coverage of hypothesized enemy routes.

An agent invokes imagery through the application of an operator (construct, transform, generate, and inspect) in Soar's procedural long-term memory (top right of Figure 6-1). We use Soar's subgoaling mechanism to implement these operators. Therefore, if the agent is in an imagery problem space and "important" information arrives (e.g., the teammate in the Scout domain sends the agent a report), the system is responsive, interrupts imagery processing, attends to the incoming input, and incorporates the new information.

Whether the task requires spatial or visual imagery, the system first must *construct* the quantitative spatial representation by accessing and combining spatial configurations from Soar's symbolic memory with general shape information in VLTM. If a depictive representation is required, imagery *generates* a visual depictive representation by combining the quantitative spatial representation from the Object Map with each object's specific shape and color from VLTM. The agent may *transform* or *inspect* the representation in the Object Map or Visual Buffer. As imagery proceeds, the symbolic results of manipulations and inspections are stored in the VS-STM and transmitted to Soar via the listeners. Although we discuss the imagery operators in a sequence, the processing is conditional and iterative.

#### *6.2.2.2.1 Symbolic Representations*

A symbolic structure in Soar is associated with a visual symbol via a *visual-object* attribute. For example, an agent's symbolic, short-term memory represents the place setting as shown in Figure 6-8 (only three objects shown). Each entity (napkin, fork, place setting, plate) has a visual symbol associated with it (V23, V9, V7, and V12 respectively). A visual symbol with a *visual-id* (napkin, fork, plate) has an underlying shape and color representation in Visual LTM. Alternatively, as in the case of the place setting, a visual symbol may have a *has-a* attribute and a spatial description specifying how it is composed of other visual symbols. Note that in addition to the visual symbols, the symbolic representation enables one to associate other, non-visual, symbols to each entity (e.g. a napkin is used to wipe your mouth, a plate holds food). A visual symbol may arise from memory retrieval, perceptual input, or after imagery instantiates an imagined object in the scene (which is the same as perceptual input from Soar's perspective). An instantiated visual symbol will have an *instance-id* corresponding to the instance-id in VS-STM (Figure 6-6) signaling that it is a salient object in the scene.

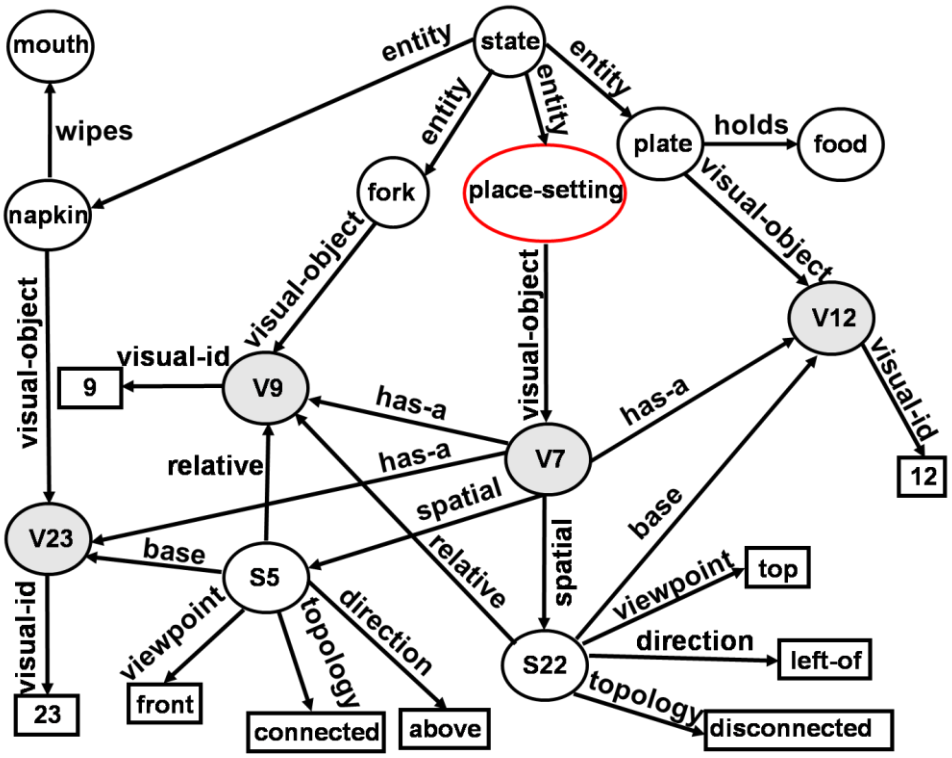


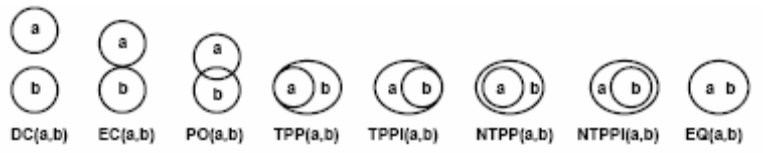

**Figure 6-8: Example Short-term Memory Symbolic Structure**

The spatial structure describing the configuration of a visual symbol includes a base visual symbol and a relative visual symbol so that a predicate relationship is defined as *<spatial-relationship> (<relative>, <base>)* (e.g. above(fork, napkin), disconnected(fork, plate)).[9] The spatial relationships may be qualitative or quantitative (Figure 6-9 and Figure 6-10). In the place setting example, the fork is above (direction) and externally connected (topology) to the napkin and left-of and disconnected from the plate. In the Scout domain, a map-icon may be located 500 meters southeast of the agent's map-icon and above and externally connected to the background (i.e. place it on top of the terrain). When there is a topological relationship, the direction specifies where the topological relationship applies (e.g., fork is externally connected to the napkin in the "above" direction). Task knowledge may rearrange the spatial relationships or even synthesize composite objects to enable the creation of novel representations (e.g. imagining an elephant on top of a house).

| Type | Qualitative | Quantitative |
|---|---|---|
| *Direction* | **left-of, right-of, in-front-of, behind, above**, below, **between**, **center-of** | **3D vector <x,y,z>** |
| *Distance* | near, far | **scalar** **or** **3D vector (distance in each direction)** |
| *Orientation* | **north, northwest, west, southwest, south, southeast, east, northeast** | **scalar** **or** **3D vector (orient towards location)** |
| *Topology (see* Figure 6-10*)* | **disconnected (DC), externally-connected (EC)**, partially-overlaps (PO), tangential-proper-part (TPP), **non-tangential-proper-part (NTPP)** | None |
| *Geometry* | **parallel**, perpendicular, **intersect**, **line-segment, triangle, angle**, **congruent** | None |
| *Size* | smaller, larger, equal | **scalar (1D length)** 2D vector (length,width) 3D vector (length,width,height) |
| *Symmetry* | **horizontal-symmetry, asymmetry, vertical-symmetry** | |

**Figure 6-9: Spatial Properties**
**Bold font indicates that the property has been used in the experimental domains**

[9]Ternary spatial relationships (e.g. between) are also possible where there are two base objects (e.g. between (plate, fork, knife)).




**DC = Disconnected**
**EC = Externally Connected**
**PO = Partially Overlaps**
**TPP = Tangential Proper Part**
**TPPI = Tangential Proper Part Inverse**
**NTPP = Non-Tangential Proper Part**
**NTPPI = Non- Tangential Proper Part Inverse**
**EQ = Equal**

**Figure 6-10: RCC-8 Topological Relationships**
**(from (Cohn et al., 1997))**

| Type | Qualitative | Quantitative |
|---|---|---|
| *Shape* | **point, vertex, line, line-segment, triangle, curve, curve-segment, enclosed-space** | **3D vector <x,y,z> (set of points)** |
| *Color* | red, green, blue, etc. | **3D RGB vector <r,g,b>** |

**Figure 6-11: Visual Features**
**Bold font indicates that the property has been used in the experimental domains**

Similar to VS-STM's visual object structure, the visual symbol in Soar's symbolic memory may have associated *visual-features* or *spatial* attributes encoded during perception, retrieved during an imagery inspection, or declared in a symbolic memory. For example, in the alphabet experiment, the initial, visual symbol for the letter 'A' may be described as being composed of three line-segments with a specified spatial configuration (i.e. one line-segment has a counterclockwise orientation of 45 degrees, another line-segment is the same size as the first line-segment but rotated clockwise 45 degrees, etc.). The agent, wanting to know if the letter A has an "enclosed space", constructs the quantitative spatial representation in the Object Map and augments the letter A's visual symbol with an instance-id. After the visual depictive representation is generated and inspected for the enclosed space, the letter A's visual symbol is augmented with a *shape* visual-feature where the shape is an enclosed space. Figure 6-11 highlights the visual features SVI currently supports.

SVI assigns a temporary *emergent-id* to shapes found during inspection so that if the agent desires more information regarding the feature, it can be retrieved from VS-STM. For example, the agent may want to know the size (i.e. length) of the letter A's enclosed space and its direction and distance from the top of the 'A'. The emergent-id supports this capability, binding the symbolic shape representation in Soar to its underlying depictive representation in the Visual Buffer via VS-STM. In a similar

manner, retrieved spatial properties that apply strictly to the letter 'A's' visual symbol (rather than the entire scene) augment the visual symbol as a *spatial* attribute (e.g. the letter 'A' enclosed space's center is above its horizontal line-segment).

#### *6.2.2.2.2 Construction*

Although the symbolic structure of the place setting in Figure 6-8 encapsulates a lot of information, it does not indicate the center object of the place setting or whether the fork is wider than the spoon—either directly or through logical inference. When there is a lack of spatial or visual knowledge relevant to the agent's current goal, an impasse occurs and Soar creates a special, imagery state where the agent directs the processing. The first step in imagery processing is to construct the desired scene. The imagery *construct* operator has two sub-commands, *compose,* and *add*. The compose command is useful when the agent is imagining a scene by initially composing two objects retrieved from memory and adding them to a "blank" scene. For example, the agent begins imagining the place setting by composing the fork and napkin. The add command is useful when adding an object to an existing perceived or imagined scene (e.g. add a knife to the right-of and disconnected from the plate, add a hypothesized enemy icon to the map relative to the existing enemy map-icon).

Functional processes within SVI respond to the specific imagery command. In the case of construction, the *Constructor* receives the operator's symbolic information, interprets it, and builds the quantitative spatial representation in the Object Map by combining each object's general shape information from Visual LTM with spatial knowledge from Soar (Figure 6-12). The symbolic information includes the visual-id and spatial properties of the object(s) being composed or added. The visual-id enables the constructor to access the object(s) general shape, or scene-graph, and instantiate it by copying the structure.[10] The spatial properties description includes a *relative-visual-id* and a *base-visual-id* or a *base-instance-id*.[11] The relative-visual-id is the visual-id of the relative object in the spatial relationship (e.g. left-of (relative-object, base-object)). If composing objects, then the *base-visual-id* identifies the base object in the spatial

[10]The color and specific shape representation (vertices) are not copied. VLTM and the Object Map share these structures and they are activated when generating a visual depiction.
[11]There is also a base-instance-id-tert for ternary spatial relationships (e.g. between).

relationship because the object has not yet been instantiated. Otherwise, a *base-instance-id*, which is the instance-id of a visual object already in the Object Map, identifies the base object.

Spatial properties may include any of the following: *direction*, *distance*, *orientation*, *size*, *topology*, and *geometry* (Figure 6-9). If the spatial properties are qualitative (e.g. left-of, northwest, externally connected), the Constructor converts the symbol to a metric representation. For example, *left-of* becomes a vector (<-1, 0, 0>) and *northwest* a scalar, absolute orientation (135 degrees). Topological information is computed from the appropriate object's bounding box or convex hull in the direction specified by the direction relationship (i.e. fork is externally connected in the "above" direction). If direction is missing then the default behavior is to create the topological relationship in a random direction. Note that this is only a "rough" topological interpretation and in order to determine a more accurate topological relationship requires constructing the "rough" topology, generating a depiction, inspecting it, projecting the coordinates from 2D to 3D space, and making translation adjustments. We have not implemented this functionality.

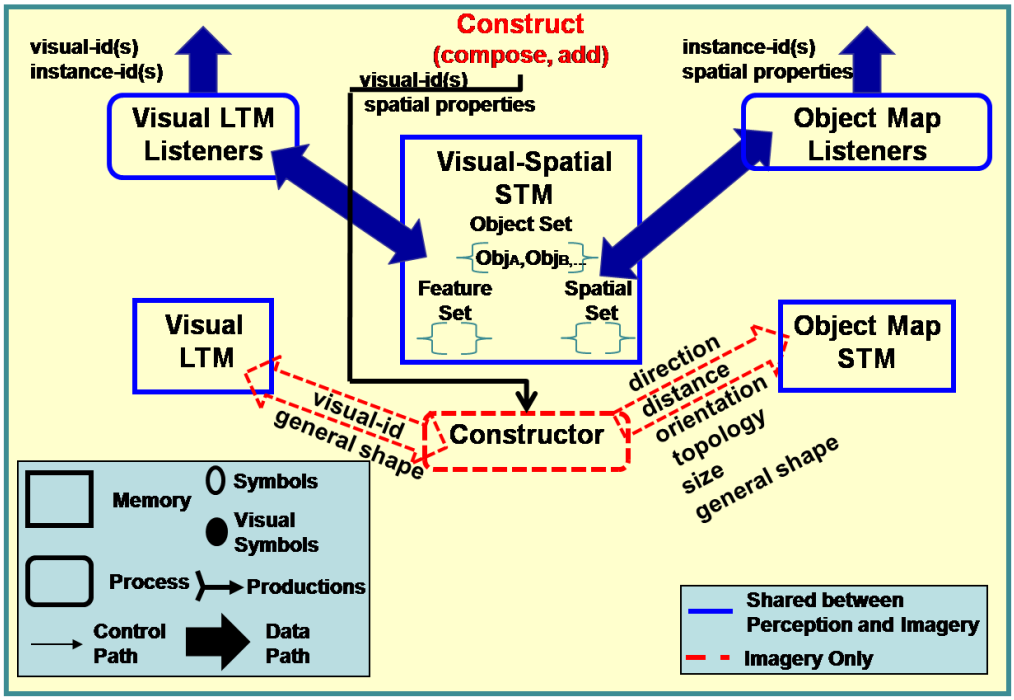



**Figure 6-12: Imagery Construction**

As an example of building a quantitative representation from qualitative symbols, consider the place setting where the agent first composes the knife and the plate by locating the knife to the right of the plate. If the construction operator does not include any distance constraints, as in this scenario, then the Constructor considers the general shape of the objects and uses a heuristic based on an object's *area of influence (Kettani & Moulin, 1999).* Locating the knife right of the plate without any distance constraints implies placing the knife next to the plate (externally connected) and then adding "a little" empty space based on the length of the objects' convex hull in the direction of the placement (right-of). In order for the Constructor to orient a visual object, the system has to assume that each object stored in VLTM has a front and a canonical orientation. SVI assumes that each object stored in VLTM is oriented "north" (e.g., the prongs of a fork face north).

After the Constructor finishes composing or adding the visual object(s) to the Object Map, feedback to Soar proceeds in a similar fashion as automatic, bottom-up visual processing. The Constructor creates a visual object symbolic structure in VS-STM for each imagined object and augments it with an instance-id, thus fulfilling the role of the Saliency Inspector and "What Inspectors." Since the object(s) added to the Object Map has already been "recognized" (that is the agent knows the object(s) it is imagining[12]), this processing is all that is necessary. In the case of the where pathway, the Constructor automatically invokes the "Where Inspectors" to inspect the Object Map (rather than the Visual Buffer) for the recently added visual object(s) direction, distance, orientation, and size information relative to the current viewpoint. As in bottom-up processing, the "Where Inspectors" add this information to the spatial set of the current scene (Figure 6-6). The VLTM Listeners and Object Map Listeners consolidate the results for input to Soar on the subsequent input cycle (Figure 6-7).

In addition to constructing objects retrieved from Visual LTM, the Constructor can also imagine or "draw" simple shapes such as those listed in Figure 6-11. This functionality is useful when the agent wants to imagine a shape it has never seen before

[12]Although the agent "recognizes" the objects added to the scene, it does not immediately recognize the resulting, composite object(s). The inspection process has to be invoked to facilitate recognition. For example, imagine a 'D' rotated 90 degrees counterclockwise on top of a 'J'; the 'D' and 'J' are immediately "recognized", but an inspection process must recognize the "umbrella".

(e.g. a view frustum). Alternatively, the shape may be the result from an inspection of the Visual Buffer that the agent wants to make a first class visual object. Making the shape a first class object facilitates spatial reasoning (e.g. determining the direction between the centers of the enclosed space and the letter A). In these cases, rather than sending a visual-id, Soar sends the vertices and indices describing the shape or its emergent-id to the Constructor.

#### *6.2.2.2.3 Generation*

If a depictive representation is required, the *generate* operator initiates processing (Figure 6-13). The *Refresher* interprets the command and combines specific shape and color from Visual LTM with the Object Map's quantitative spatial representation to generate the visual depictive representation in the Visual Buffer. Generation may render all or some of the visual objects and, as previously discussed, create more than one image, or vb-layer, as a form of cognitive focus (see section 6.2.1.1). The agent may optionally specify a "generation color" for a visual-object to distinguish it as a unique object in the image. After the image is generated, the Refresher updates the current scene's visual object set in VS-STM with the generated vb-layer and its associated visual objects along with their optional generation color. This information is transmitted to Soar via the Visual LTM Listener during the next input-cycle.

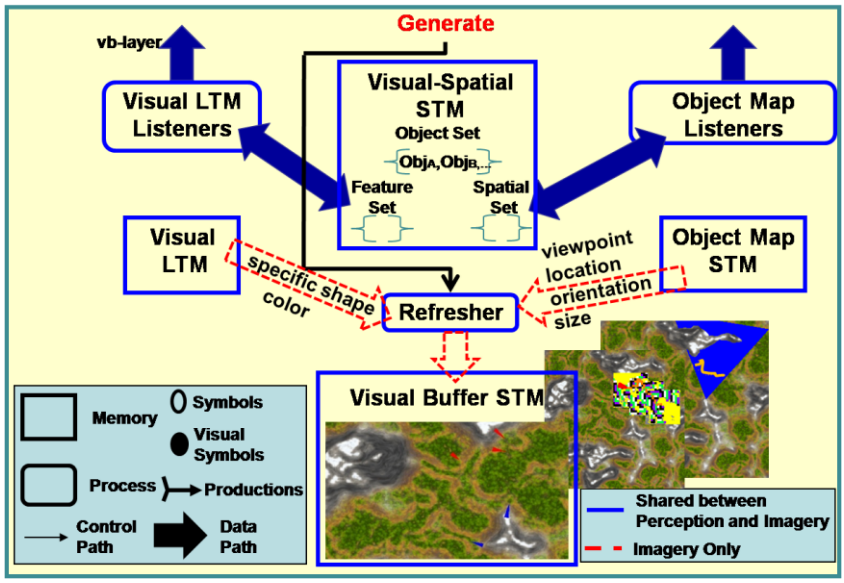



**Figure 6-13: Imagery Generation**

#### *6.2.2.2.4 Transform*

The *transform* operator manipulates the Object Map's quantitative or a Visual Buffer's depictive representation through their respective *Manipulator* processes (Figure 6-14). Manipulation of the Object Map includes transforming (i.e. translation, rotation, scaling) a specified object in the scene or changing the viewpoint. The modification of the viewpoint enables the agent to change its perceptually based egocentric view to an imagined allocentric view in order to infer new spatial relationships. For visual imagery, viewpoint manipulations can serve as an attention mechanism where, prior to generating an image in the Visual Buffer, the system can transform the viewpoint to another perspective, focus the viewpoint in or out, or shift it in any direction. These transformations effectively focus attention on specific areas of the Object Map so that when the Refresher renders the scene to the Visual Buffer the generated pixels are representative of that viewpoint. Although we have implemented this viewpoint transformation, we have only used it for changing an agent's "frontal" view to a top-down "map" view and for debugging.

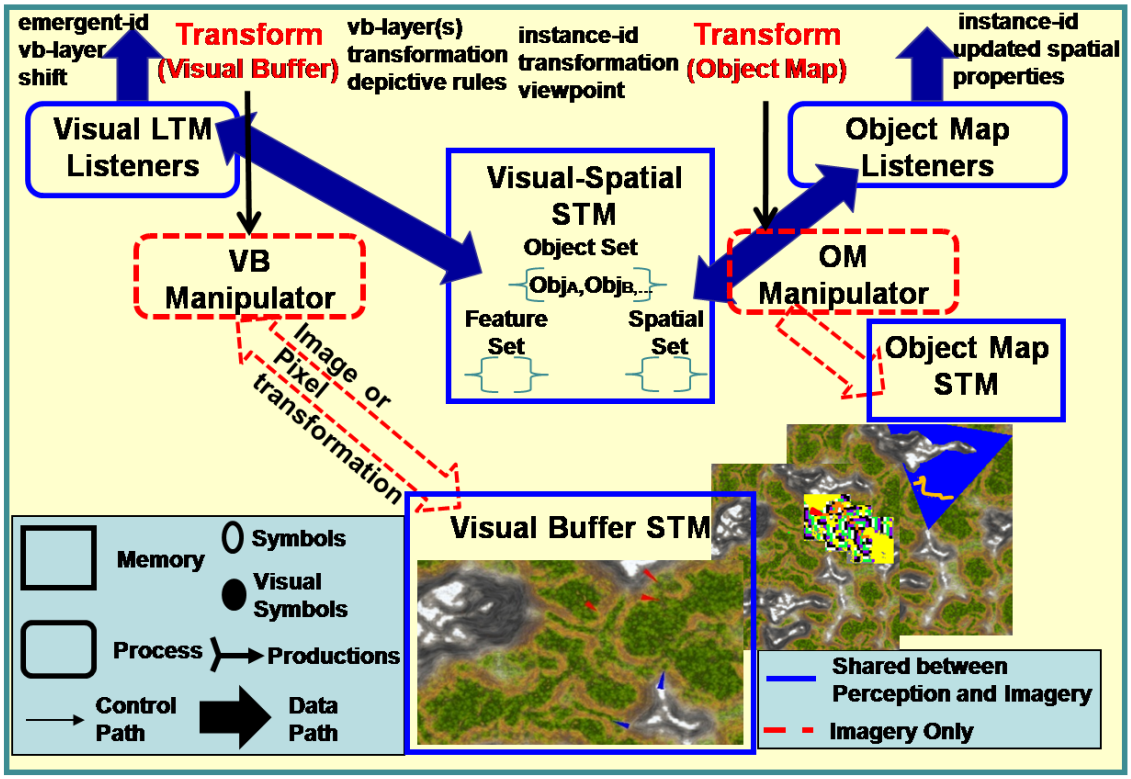



**Figure 6-14: Imagery Transformation**

In the case of transforming a visual object node in the Object Map, the transform operator sends the instance-id and desired transformation to the *Object Map Manipulator*. The Object Map Manipulator accesses the visual object's scene graph node by looking up its instance-id in VS-STM (Figure 6-6). It then modifies the local transformation of the node (Figure 6-4) and recursively traverses children of the node to update their world transformations (i.e. if you transform a car you want all of its parts to move with it). For example, in the scout domain, the agent modifies the orientation of the teammate's and its own imagined views to determine if they provide better coverage of possible enemy routes. As the agent receives reports from its teammate or visually observes changes in an entity's location or orientation it transforms the associated map-icon. When the agent or its teammate loses visual contact with an enemy, it can imagine simulated movement based on task knowledge of an enemy vehicle's velocity. Note that the "simulation" in this case is a one-step process. Future work will discuss transformations that use motion models of a particular entity (such as a tank) to simulate movement over time.

After the Object Map Manipulator finishes transforming a visual object, feedback to Soar proceeds in a similar fashion as described for the Constructor. The Object Map Manipulator automatically invokes the "Where Inspectors" to inspect the Object Map for the modified visual object. The inspectors update the spatial information in the current scene's spatial set (Figure 6-6), and the Object Map Listeners consolidate the results for input to Soar (Figure 6-7).

For Visual Buffer manipulations, the *VBManipulator* receives a vb-layer identifier and either a transformation command or a set of *depictive rules* from the *transform* operator and manipulates the image(s) corresponding to the vb-layer(s) identifier (Figure 6-14). Image transformations include standard image processing techniques (e.g. rotation, scaling, and kernel filters). Depictive manipulations are based on the pixel-level rewrite system (Furnas, 1990, 1991; Furnas et al., 2000) and discussed in depth, to include examples from the Alphabet experiment and Scout domain, in Appendix B. We briefly summarize the details here.

Unlike sentential, mathematical-based processing such as Gaussian filters or the Hough transform (Appendix B), pixel-level rewrites take advantage of the topological structure and color of a depictive representation. Similar to a production system, there are

a set of rules with a left-hand side (LHS) and a right-hand side (RHS), but rather than predicate symbols, the LHS conditions and RHS actions are visual depictive representations that operate on a shared image. The color and shape of each LHS depiction, determines a match rather than the syntactic structure of the symbols.

Figure 6-15 illustrates an example of two depictive rules. The top rule is a 1x2 rule stating, "if there is a black pixel adjacent to a gray pixel then change the gray pixel to a white pixel." Similarly, the bottom rule is a 2x2 rule that says, "if there is a black pixel diagonally adjacent to a gray pixel then change the gray pixel to a white pixel." The asterisks represent wildcard values, and a rule may specify alternate rotation orientations (90, 180, 270 degrees) for matching. While there are rule matches, the processing iterates over the image. When a rule matches a region of the image, the RHS action rewrites the appropriate pixel(s). Each rule has a priority associated with it to facilitate sequencing, so if two or more rules match, then the rule with the highest priority fires. Although the matching and modifications are local in nature, the cumulative effects have global consequences.

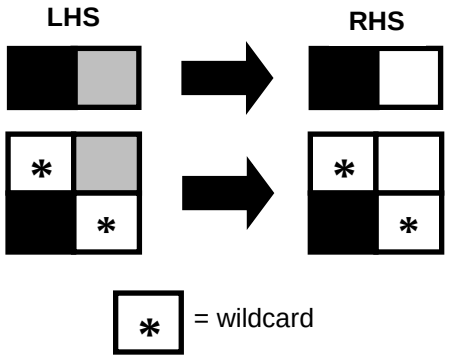



**Figure 6-15: Example Pixel-level Rewrite Rules**

We made three primary extensions to the pixel rewrite system to support processing in the Soar+SVI architecture. First, the *depictive rules* are encoded in Soar's production memory as a set of transform operator elaborations.[13] If the operator is selected, the rules are added to Soar's output-link and sent to the VBManipulator. The VBManipulator receives the rules and executes the pattern matching over the image specified by the operator's vb-layer(s).

[13]An operator elaboration is a type of Soar production or rule.

Second, we have a notion of a *rule-set* and make a distinction between three types of depictive rules: *threshold*, *pattern*, and *mark*. A rule-set contains a set of rules of the same type (threshold, pattern, or mark). Similar to rules, a rule-set has an associated priority to enable sequencing among rule-sets. Processing of *threshold* rules makes one pass through the image, and, for each pixel, changes the value based on an exact match, a minimum, a maximum, or a range of pixel values. In the Scout domain, we found this rule type useful for marking known obstacles such as buildings and "no-go" terrain. The functionality of the *pattern* rule is exactly as the pixel-rewrite system where the processing iterates over the image matching and firing rules until there are no more rule matches. An agent uses this type of rule for determining enclosed spaces in the alphabet experiment and creating a distance field flood in the Scout domain.

We chose to distinguish the *mark* rule type as a signal to the VBManipulator that it should create a shape object for the marked region in VS-STM. Prior to processing the mark rules, the VBManipulator instantiates a shape object and adds it to the current scene's Visual Feature Set (or the appropriate Visual Object) in VS-STM (Figure 6-6). The shape object includes an emergent-id, the marking color, and the set of points marked during the processing. The subsequent Soar input cycle creates a symbolic shape structure in Soar's STM and records the emergent-id. The points remain associated with the shape in VS-STM to support inspection (e.g. what is the length of the shape?) and possibly construction (e.g. add the shape as a first-class visual object to the Object Map).

Although we could achieve the same functionality with the pattern rule type, we also chose to distinguish the *mark* rule type for efficiency purposes.[14] The VBManipulator processes *mark* rules in a similar fashion as the pattern rules, but the processing starts at a location specified in the transform operator and proceeds in the "active" direction based on the rule's RHS. Mark rules are either 1x2 or 2x2 diagonal rules where the RHS pixel rewrite is always the center pixel and the other active (non-wildcard) cell determines the next processing direction. If a rule has more than one active match (i.e. by a rotation of the LHS), then the VBManipulator records other matching pixel locations by pushing them on a stack. After the VBManipulator exhausts processing in the chosen direction, the pixel locations on the stack are popped and, if not already

[14]This type of processing may also be inherent to the pixel-rewrite system.

marked,[15] processed. We found the mark set of rules useful for recording the hypothesized enemy paths in the Scout domain and propose that this form of processing is useful for marking salient objects by the Saliency Inspector during bottom up visual processing.

The final extension we made to incorporate the pixel-rewrite system is to create an *attention window* (Kosslyn, Thompson, & Ganis, 2006) in order to keep the image size manageable for computational efficiency. One of our computational constraints discussed in Chapter 3 is that the processing of the representation must be efficient so that the agent remains reactive to changes in the environment. In our implementation, the attention window is a fixed, m x m region, where m is a multiple of two. The size of the attention window and its shift direction is task knowledge transmitted from Soar to the VBManipulator every decision cycle in which the manipulation is active. Note that this is a shift of the cognitive focus and not a shift of the agent's "eyes."

#### *6.2.2.2.5 Inspection*

After the system has constructed, transformed, and, if necessary, generated the depictive representation, conditions are set for the inspection process (Figure 6-16). Inspection may focus on the Object Map (spatial imagery) or the Visual Buffer (visual imagery), and the appropriate "What" or "Where" inspectors process the representations based on the query parameters. The *inspect* operator provides the symbolic query. For example, "what is the center object of the place setting?", "what is the orientation angle between line-A and line-B?", "does the letter 'O' have a curve?", or "how much of the teammate's view covers enemy-1's hypothesized path?"

An *Inspector* process (not shown in Figure 6-16) intercepts the command from Soar and dispatches the appropriate What or Where inspector based on the query type. The query types are the same as the spatial and visual properties listed in Figure 6-9 - Figure 6-11. Each query type has an associated inspector. For example, the "Where Inspectors" include a *DirectionDistanceInspector*, *OrientationInspector*, *TopologyInspector*, *GeometryInspector*, etc. and the "What Inspectors" include a *LineInspector*, *CurveInspector*, *ShapeInspector*, etc. Each inspector maintains a reference

[15]For example, when processing a closed region, the processing returns to the starting/ending point.

to the master Inspector so if a query involves multiple parts (e.g. direction and topology) the initial inspector handling the query can invoke another inspector as necessary. Each inspector also maintains a reference to the Visual Buffer Manipulator in the case a transformation of the Visual Buffer (e.g. detecting enclosed spaces) facilitates the inspection. As with bottom-up visual processing, the results of the inspection(s) are stored in the appropriate visual feature or spatial set in VS-STM, and the listener processes consolidate the results for Soar's symbolic memories.

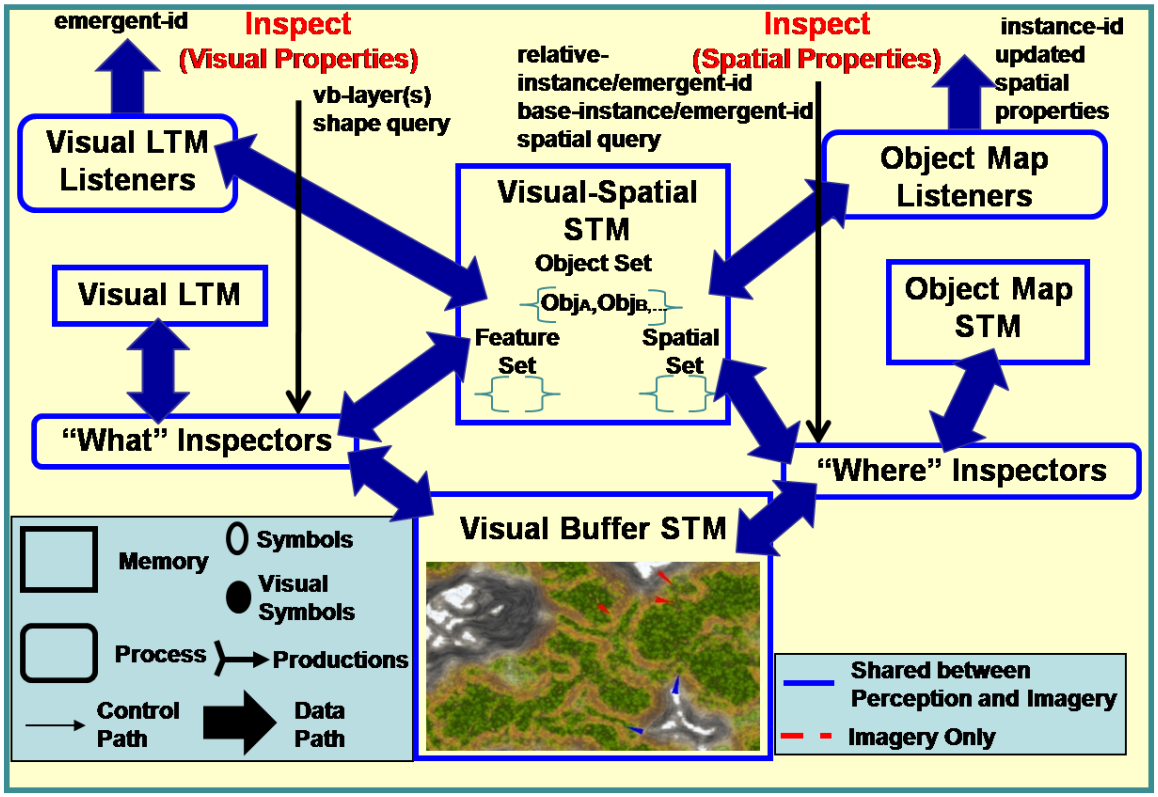



**Figure 6-16: Imagery Inspection**

Spatial imagery entails queries of visual objects in the Object Map or comparisons between emergent shapes in VS-STM with visual-objects. "Where Inspectors" receive at a minimum a *relative-instance-id* and possibly a *base-instance-id*[16](Figure 6-16). When there is only a relative-instance-id and the spatial query is specified in qualitative terms, the inspector's behavior is to return all the visual-objects in the current scene meeting the constraints of the query (e.g. center-of (place setting), intersect (line-A), externally-connected (napkin), southeast (agent-map-icon)). In the case where both the relative- and

[16]Again, ternary spatial queries (e.g. between) include an additional base-instance-id.

base-instance-ids are provided, the inspection is an assertion of the truth (e.g. in-front-of (triangle, line-A), northeast (enemy-2-map-icon, agent-map-icon)). An agent signals a request for quantitative results by including an empty *vector* or *scalar* attribute (e.g. orientation (fork, *scala*r), intersection (line-A, line-C, *vector*), direction (enemy-1-map-icon,key-terrain-A,*vector*)). For geometric and topological relationships, binary queries may optionally specify the creation of an emergent shape object that satisfies the geometry or topological relationship (e.g. intersect (Line-A, Line-B, create-vertex)).

Visual imagery requires inspection of the Visual Buffer for spatial or visual properties. The appropriate inspector receives a *vb-layer(s)*, a *relative–instance-id* (visual object), or *relative-emergent-id* (retrieved shape) and possibly a *base-instance-id* or *base–emergent-id* (Figure 6-16). The vb-layer(s) specifies the set of images involved in the inspection process for shape features. The instance- or emergent-ids give the inspector direct access to the visual-object or retrieved shape objects stored in VS-STM. As previously discussed, these symbolic structures may contain the object's marking or generation color and associated vb-layer(s) in which the object appears. The inspector uses this information to perform the desired inspection. For example, in the Scout domain, the visual depictive representation of the teammate's view is in one image, and each imagined path (three, one for each enemy/key-terrain pair) is in a separate image. Each image has an associated vb-layer identifier, each marked path has an associated emergent-id, and the teammate's view has an instance-id as it is an imagined visual-object.

Visual imagery inspections for visual-features (Figure 6-11) are implemented in qualitative, unary terms only (e.g. curve(letter-A), enclosed-space (letter-C)). If the appropriate inspector finds the feature, it creates an emergent shape object and stores it in the appropriate visual-feature set in VS-STM. The Visual LTM listener transmits the emergent-id of the shape to Soar. Inspection of spatial properties in the Visual Buffer are similar to spatial queries involving visual objects in the Object Map as the query may be qualitative or quantitative, and the number of instance- or emergent-ids in the query parameters determine the type of results the inspector returns. As with spatial imagery, unary queries return all visual objects (instance-ids) or shapes (emergent-ids) in the vb-

layer(s) image satisfying the constraints. Binary or tertiary qualitative queries return a truth assertion.

Similar to spatial imagery, for geometric and topological relationships, binary queries may optionally specify the creation of an emergent shape object that satisfies the geometry or topological relationship. For example, in the Scout domain the agent may ask the inspector to determine the subset of a hypothesized enemy path shape that satisfies the non-tangential-proper-part topological relationship with the teammate's view frustum (e.g. non-tangential-proper-part (path-1, teammate-view, create-shape)). For quantitative queries (e.g. size-of (path-1,scalar)), the inspector determines the specified scalar or vector value in the two-dimensional image space and converts the metric information into a three-dimensional space based on the current location and direction of the viewpoint. This conversion insures that the reasoning from Soar's perspective is based on the Object Map's three-dimensional Euclidean space.

#### 6.2.2.3 Synchronization of Perception, Imagery, and Cognition

As discussed in section 6.2.2.1, bottom-up perceptual processing proceeds along two simultaneous pathways while operators in Soar's procedural memory, executing in a third processing path, attend to the listeners' input (Figure 6-7). Imagery processing initially deviates from these processing paths in that it originates from the application of an imagery operator in Soar and flows to the corresponding imagery process. This processing diverges into five possible paths to include construction, manipulation, generation, and inspection (two parallel paths). Figure 6-17 shows four paths emanating from Soar's output as we have combined the construct/manipulate path and, as in bottom-up visual processing, the "What" and "Where" inspectors could process in parallel. The imagery process performs its function (i.e. construction, transformation, generation, inspection), and returns to the Soar decision cycle path where consolidated input from SVI's listeners is processed. Although the processing path is different from bottom-up, visual processing, the information (i.e. data) flows along the same paths. For example, after processing constructs or manipulates a representation, the information (e.g. the instance-id of the imagined/transformed visual object, automatically extracted spatial relationships from the current viewpoint) resulting from this construction/manipulation

are extracted by the inspectors, stored in VS-STM, and consolidated by the listeners for input to Soar. During imagery inspection, as in bottom-up perception, the inspectors extract information from the quantitative or depictive representation and encode it in VS-STM. The listeners consolidate the results from VS-STM for input to Soar.

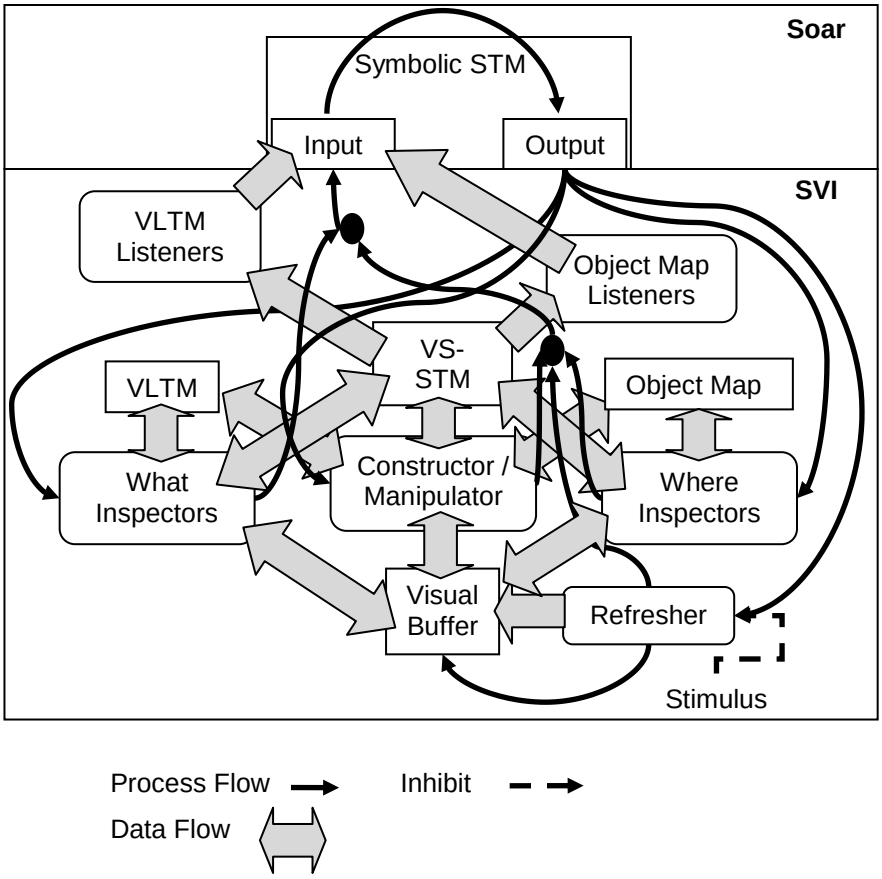



**Figure 6-17: "Top-down" Imagery Processing and Data Flow**

This top-down imagery processing propagates temporal and spatial constraints on the overall system. Imagery may inhibit incoming sensory input (bottom of Figure 6-17) and synchronize (i.e. "lock") the memories to avoid race conditions. That is, sensory information from the environment continues to arrive, but in order to give imagery an opportunity to process the spatial and visual representations, there may be an ephemeral inhibition of sensations. Of course, such inhibitions cause the system to miss incoming information so there must be some sort of an "override" to interrupt imagery and attend to the incoming sensory input. Although we do not have an adequate model of how such an "override" is realized, for now we say that the system must be responsive to

"important" information. For example, in the Scout domain "important" information is a moving object in the perceived scene or a message arriving from the agent's teammate.

An orthogonal issue relevant to the processing flow of imagery and inhibition of sensory input is the system's overall truth maintenance mechanisms. That is, the architecture must address how the information states between sensory input, imagery, and cognition remain consistent. For example, if there are two salient, visual objects in the perceived or imagined scene, then that information must be reflected in at least one of the Visual Buffer's images, in the Object Map, in VS-STM, and on Soar's symbolic, input-link. Similarly, if a previously imagined object is no longer in the current scene because the agent is "perceiving" again, then the visual symbol instance in Soar's short-term memory must be tagged and eventually removed so that future reasoning does not assume the object is still in the scene.[17] In most cases, the VLTM and Object Map Listeners will handle the updating of symbols on Soar's input-link, but if the agent creates any internal symbolic state in Soar, it has the responsibility through procedural knowledge to enforce consistency.

From an architectural perspective, after imagery finishes processing, or when an "important" perception interrupts it, any temporary imagery state stored in the short-term memories is removed. For example, in the Scout domain if an observation report from the agent's teammate arrives while the agent is imagining a hypothesized enemy path, the architecture removes (1) the symbolic state created in Soar's imagery problem space,[18] (2) any imagined visual objects in the Object Map, (3) temporary images created in the Visual Buffer, and (3) any state containing imagined objects in VS-STM. It then updates those short-term spatial and visual memories based on the current visual perception. The effect then is that any state imagery creates is monotonic with respect to any pre-existing state. The agent, through procedural task knowledge, must decide what information to make persistent in one of Soar's symbolic memories and is responsible for enforcing consistency based on the spatial and visual information arriving on Soar's input-link.

[17]Future work discusses the exploration of "object permanence" or the memory of a visible object that disappears (through motion).
[18]The state for the imagery problem space is created by Soar's subgoaling mechanism.

## 6.3 Summary

In summary, the architecture consists of two major components, Soar and SVI. Each contains a set of fixed memories and processes with Soar encompassing the symbolic representation and SVI including the quantitative spatial and visual depictive representations. A symbolic VS-STM binds the two processing pathways and serves as a temporary symbolic store. Imagery processing leverages the mechanisms high-level vision inherently provides and includes functions for constructing a quantitative spatial representation, generating a visual depictive representation, and transforming or inspecting either representation. The next chapter will focus on the evaluation of this architecture.

# Chapter 7

# Evaluation

This chapter presents the objective evaluation and continues the subjective evaluation of the architecture in three different domains. The objective evaluation is in support of our research goal to understand the computational capabilities of spatial and visual imagery. It focuses on three metrics: efficiency, functional capability, and problem-solving quality. As there are no existing cognitive architectures with an imagery component and corresponding experiments to compare our system against, the objective evaluation includes, where possible, comparisons between Soar agents with and without the Spatial-Visual Imagery (SVI) component. This approach supports our research goal by providing quantitative evidence that the capability and computational gain an agent achieves is a result of spatial and visual imagery mechanisms and not task knowledge.

The architectural description in the previous chapter is the start of our subjective evaluation. In this chapter, we will expand on that discussion as it relates to the computational constraints introduced in Chapter 3 (Figure 3-1). Although our primary goal focuses on functionality, we briefly touch on behavioral constraints when we present results from the Alphabet experiment. These results make comparisons with human data and highlight shortcomings in the architecture. These shortcomings include our uncertainty of the visual processing algorithms that humans use to recognize features, and the architecture's lack of an "image maintenance" mechanism that regenerates the image when it decays (Kosslyn, Thompson, & Ganis, 2006).

The remainder of this chapter is organized as follows. We begin by presenting the evaluation criteria. We then provide, for each experimental domain, additional implementation details, relevant results for applicable criteria, and our subjective assessment. We present each experiment in chronological order corresponding to the evolution of the architecture. As such, the first two experiments do not always have a one-to-one mapping with the current architecture as we apply the lessons learned from earlier experiments to improve the architecture. We conclude the chapter with the major lessons learned.

## 7.1 Evaluation Criteria

This quote from Pylyshn (2002) serves as inspiration as to what we are attempting to achieve.

> "The search for a system of representation that retains some of the attractive features of pictures and yet can serve as the basis for reasoning has been the holy grail of many research programs, both in cognitive science and in artificial intelligence."

Figure 7-1 illustrates our evaluation criteria. There are four dimensions. The first dimension is the behavioral, biological, functional, and computational constraints influencing the architectural design space that we must continuously consider. The x-axis represents computational gains, the y-axis represents additional functional capability, and the z-axis is an indicator of the problem-solving quality. In the foreground of these evaluation dimensions is an assessment of the task and environments where spatial and visual imagery is useful (Chapter 5).

As the architectural design considers the behavioral, biological, and functional constraints, our evaluation in this chapter focuses on the computational constraints as they relate to the other three dimensions. Specifically, we are concerned with the relative efficiency of a representation for a given task when compared to using another representation for the same task. We also want to analyze subjectively whether we can assign credit for the resulting efficiency to the architecture. Finally, even though the resulting representation may provide computational efficiency, we want to evaluate its impact on the ability of the architecture to support reactive and deliberate behavior.

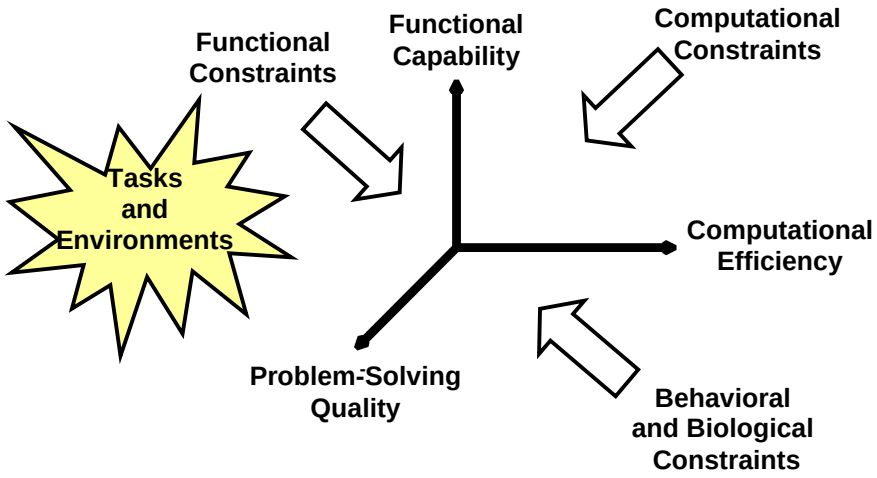



**Figure 7-1: Evaluation Metrics**

We evaluate computational efficiency by comparing either different designs or different agents with and without SVI. Agents using SVI are denoted Soar+SVI (Soar "plus" SVI) while agents not using SVI are denoted Soar-SVI (Soar "minus" SVI). The comparisons include the amount of processing (in CPU time) and the amount of task knowledge in terms of Soar productions. In order to nullify any system or simulation variability, each result represents an average of 30 trials. Where appropriate, we also discuss opportunities for improved efficiency and generality.

We measure functional capability with subjective observations derived from implementation of the task. Objective measurements of functionality compare an agent with and without imagery. Problem-solving quality measures whether the agent's overall performance improves with imagery processing. We evaluate this metric in the Scout domain. Finally, where appropriate, we highlight the tradeoffs in terms of design complexity, generality, and psychological validity.

## 7.2 Geometry Gymnastics

The first domain derives from Larkin and Simon's (1987) work demonstrating a computational advantage of a diagram. In this problem, the agent must locate objects (e.g. vertices, line segments, triangles) and infer relationships (e.g. angles, congruency) that initial task knowledge does not specify (Figure 7-2). We chose this task because it

stresses the construction and inspection of a quantitative spatial representation. As either symbolic or metric representations are sufficient, we can compare agents and determine computational and functional differences. The task does not require a visual depiction as initial knowledge specifies the objects (i.e. lines) from which other features can be directly inferred.

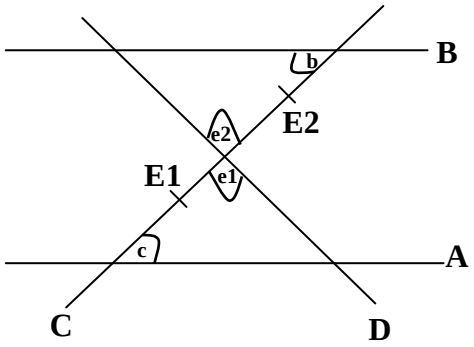



**Figure 7-2: Geometry Problem**

To review, the agent's goal is to prove two triangles formed by a given specification are congruent (Figure 7-2). The agent's initial knowledge is that there are four lines (A, B, C, and D). Line A is parallel to line B, line C intersects line A, and line D intersects line C at the midpoint of the line segment formed by the intersection of line C with line A and line B. To prove the two triangles are congruent, the agent employs the angle-side-angle (ASA) rule. The rule states that if two angles and the included side of one triangle are congruent to two angles and the included side of another triangle, the triangles are congruent. In Figure 7-2, if the agent shows E1=E2, c=b, and e1=e2, then it proves the triangles are congruent.

To solve the problem with strictly symbolic representations (Soar-SVI), an agent's task knowledge must include rules for creating the different features (e.g. vertices, line segments, triangles) and spatial properties (e.g. angles, congruent). We implement the construction of these properties using Soar operators. Figure 7-3a provides an example of some of these operators. During each decision cycle, the Soar-SVI agent progresses towards its goal by creating the features required to solve the problem. For example, operator three creates a line-segment given that there are two vertices on the same line and production six states that alternate interior angles are congruent. The

italicized operators are the key distinction between the two implementations (discussed shortly).

| (a) Soar-SVI | (b) Soar+SVI |
|---|---|
| (1) If two lines (L-1,L-2) are parallel, and L-1 intersects line, L-3, then L-2 intersects L-3<br>(2) If two lines (L-1, L-2) intersect, create a vertex, v-1, recording the two lines associated with the vertex.<br>(3) If there is a vertex, v-1, on line, L-1, and another vertex, v-2 on line, L-1, then create a line-segment, ls-1, and record the two vertices associated with it.<br>(4) If there are three line segments (ls-1,ls-2,ls-3) and ls-1shares a vertex, v-1, with ls-2, and ls-2 shares a vertex, v-2, with ls-3, and ls-3 shares a vertex, v-3, with ls-1 and v-1 != v-2 != v-3 then create a triangle, t-1, and record the line-segments that make up each side<br>*(5) If there is a vertex, v-1, create 4 angles, 1 per region. Region 1, 2, 3, 4 (see Figure 7-4)*<br>(6) If there are two vertices, v-1,v-2, and v- and v-1 is associated with line L-1 and L-2 and v-2 is associated with line L-1 and L-3, and L-2 is parallel to L-3 and v-1 has an angle, a-1 in region 1 and v-2 has an angle, a-2 in region 3 then a-1 is congruent to a-2 (Alternate interior angles congruent)<br>(7) If there is a vertex, v-,1 with angles a-1, a-2, and a-1 is in region 1 and a-2 is in region 3 then a-1 is congruent to a-2 (Vertical angles congruent**)**<br>(8) etc.<br>(9) Prove congruent with ASA rule<br><br>**54 Total Task Productions** | (1) If two lines (L-1,L-2) are parallel then Compose( L-1, L-2, geometry, parallel)<br>(2) If scene has two parallel lines(L-1,L-2) and another line, L-3 intersects L-1 then Add (L-3,L-1,L-2,direction.in-between,orientation.scalar random(30,60), size(L-1,2))<br>(3) If scene has lines, L-1, L-2, and L-3, and line, L-4, intersects line L-3, then Add(L-4, L-3 orientation (random(30,60),size(L-3,1)))<br>(4) If scene has four lines, L-1,L-2,L-3,L-4, then Inspect(intersect(scene, create-vertex))<br>(5) If scene has vertices then Inspect (line-segment (scene), create-line-segment)<br>Inspect (triangle (scene), create-triangle)<br>*(6) If triangle, t-1 with line-segment, ls-1 then Inspect(angle(center-of(t-1, ls-1),scalar)*<br>(7) *If vertex, v-1, on line L-1 and L-2 with orientation angles to triangle, t-1, center a-1 and a-2 respectively then create angle in region sign(a-1)/sign(a-2)*<br>(8) Alternate interior angles congruent<br>(9) Vertical angles congruent<br>(10) Prove congruent with ASA rule<br><br>**28 Total Task Productions**<br><br>This number does not include imagery operators (*construct, transform, generate, inspect*) that we consider "architectural". |

**Figure 7-3: Example Geometry Problem Operators (Pseudocode)**

The Soar agent using SVI (Soar+SVI) constructs and inspects the quantitative spatial representation with operators that invoke imagery (Figure 7-3b). When fully implemented the Soar-SVI agent requires 54 task productions, and the Soar+SVI agent requires 28 total task productions (not including imagery operators). Since the visual objects in this example are lines, there are two different ways we can model the agent

imagining the lines. One way is to have the agent simply "draw" each line by sending SVI a pair of vertices for each line. This model assumes that the agent has task knowledge defining the spatial arrangement of vertices for parallel lines, intersecting lines, etc.

Another more general way and what we chose to model here, is to assume that the agent has a prototypical representation of a line in VLTM. Recall that each visual object stored in VLTM has a "front" and canonical orientation. In this domain, we assume that a line's canonical orientation is due "east" (i.e. an infinite ray extending east). The Soar+SVI agent's first operator (Figure 7-3b) states that if there are two lines with a parallel relationship, then imagine composing them in a parallel manner. Note that since both of the lines face "east," this is the same as saying compose the two objects with the relative visual object left-of (or right-of) the base visual object. The objects' area of influence, which is slightly larger than the length of their bounds, is the default distance between the two objects (i.e. lines).

Continuing in a similar manner, operators two and three add lines C and D to the spatial representation. The second operator says to add line C in between line A and line B, oriented some random direction between 30 and 60 degrees and with a size twice as large as line A. The orientation and size specifications are required to form the intersection specified by the task instructions. Note that when line D is added to the spatial representation, it is added relative to line C meaning the origins of each line will be the same—thus the bisect relationship constraint is met. Line D's orientation is relative to line C's orientation so the end effect is that Line D is orientated between 60 and 120 degrees relative to line A.

Once construction is complete, the agent simply has to inspect the representation (taking advantage of the metric representation and analytical geometry) for the information it is seeking. In this problem, it requires the intersection points, or vertices between the lines (Figure 7-3b operators 4 and 5). Once it knows the vertices, it can infer the line-segments (through SVI) by determining the pairs of vertices that fall on the same line (again through analytical geometry), and find the triangles from the set of vertices and line segments (edges). This knowledge, together with the information of which side

the triangle's center is relative to each line (explained next), is sufficient to prove the congruency of the triangles with the angle-side-angle rule.

### 7.2.1 Functional Capability

One of the significant results from this experiment is that spatial imagery provides the agent with a "sense of direction" enabling it to reduce its search space. The Soar-SVI agent cannot determine from the qualitative symbolic knowledge alone, where the center of each triangle is relative to each vertex because it does not have metric information or a notion of space. Therefore, it has to create four angles for every vertex, labeling them as the angle belonging to region 1, 2, 3, or 4 (Figure 7-3a production 5 and Figure 7-4). This leads to an explosion in the problem space as, in addition to the extra memory structures required for each angle, the agent has to reason about several congruency relationships between angles that are not inherent to the triangles (e.g. Figure 7-3a productions 6 and 7). In fact, to solve the problem, the Soar-SVI agent must have specific task knowledge to apply the angle-side-angle rule (e.g. if there is a vertex representing the intersection of line A with line C, then the angle belonging to the triangle is in region 4).

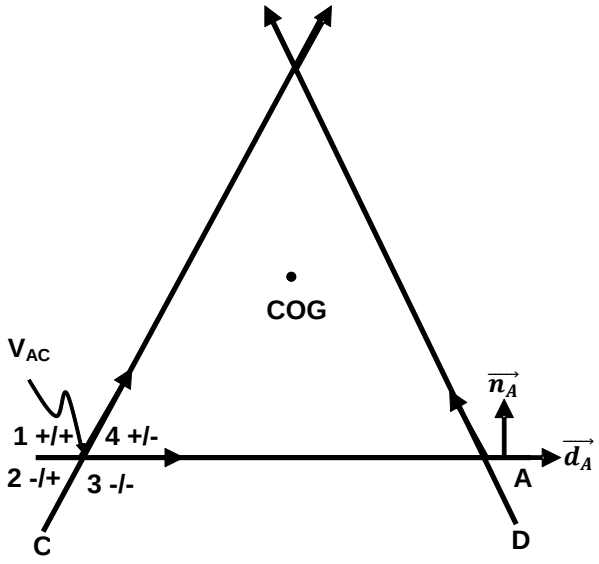



**Figure 7-4: "Sense of Direction"**

The Soar+SVI agent, however, can take advantage of the canonical direction of each line and the metric information of the spatial representation to determine the angle belonging to each triangle vertex. One approach is for the agent to query SVI for the

orientation angle between each triangle's line segment and its center of gravity (COG) to determine if the triangle is on the positive or negative side of the line segment (Figure 7-3b production 6 and Figure 7-4). SVI computes the COG as an average of the triangle's vertices. To determine the angle between the line segment and the COG, and the angle's corresponding direction (positive/counterclockwise or negative/clockwise) requires two computations:

$$\theta_{\vec{d}} = \cos^{-1}\left(\vec{d}_{ls} \cdot COG\right) = \textit{angle between line direction and } COG \quad (1)$$

$$\theta_{\vec{n}} = \cos^{-1}(\vec{n}_{ls} \cdot COG) = \textit{angle between line normal and } COG \quad (2)$$

$$\theta_{\vec{d}}\ sign = \begin{cases} 0 < \theta_{\vec{n}} \leq 90 & +/CCW \\ 90 < \theta_{\vec{n}} \leq 180 & -/CW \end{cases}$$

The first computation determines the angle between the line-segment and COG while the second computation infers the direction of this angle. If the resulting value of the second computation is between 0 and 90 degrees, then the triangle is on the positive side of the line segment (i.e. the line segment must be rotated in a counterclockwise direction to "hit" the COG). Otherwise, the triangle's COG is on the negative side of the line segment. With this information, the agent can infer the angle associated with each vertex. For example, in Figure 7-4, the triangle is on the positive side of line A and the negative side of line C. Therefore, the angle associated with vertex, $v_{AC}$, is in region four (+/-). With this sense of direction, the agent only creates symbols for angles associated with each triangle.

## 7.2.2 Computational Advantage

The functional advantage gained from the "sense of direction" directly influences the computational gain the agent achieves. The Soar+SVI agent requires less real and simulated computational time than Soar-SVI (Figure 7-5). The primary reason is the number of angles the Soar-SVI agent creates and the subsequent comparisons for congruency. For example, Soar-SVI creates 5 vertices x 4 angles = 20 angles compared to Soar+SVI's 6 angles (three per triangle). Subsequently Soar-SVI annotates congruency relationships for 24 vertical angles (i.e. 5 vertices x 4 vertical angles plus 2 pairs of vertices with 2 congruent alternate interior angles). This number does not reflect the additional congruency relationships that the agent has to consider between pairs of

unnecessary angles (Figure 7-3a operators 6 and 7). In addition to the performance gain, the Soar+SVI model also requires less task knowledge as it only requires 28 productions compared to Soar-SVI's 54 productions. The Soar-SVI agent requires knowledge about geometric structures inherent to SVI's spatial imagery processing. Functionally, this suggests that SVI decreases the amount of knowledge required to learn such a task.

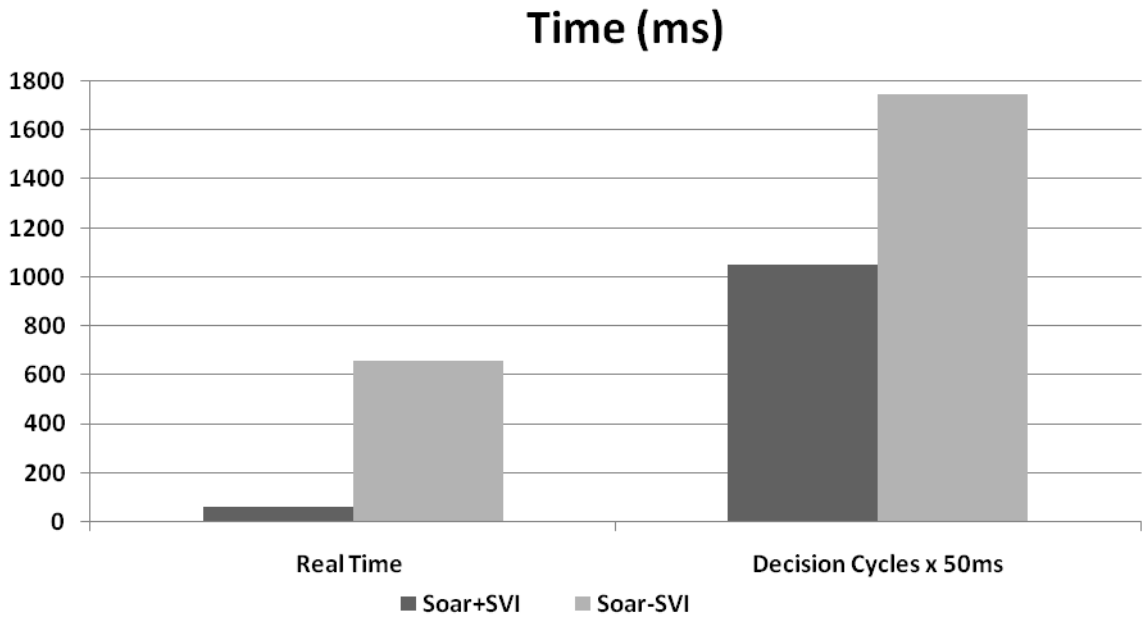



**Figure 7-5: Empirical Time for Each Agent**
**Simulated time is decision cycles x 50 ms**

To measure whether the architecture supports reactive and deliberate behavior (i.e. is responsive to the environment), we estimate the amount of time an average imagery operation requires normalized to a hypothesized decision cycle time of 50 milliseconds. We calculate the average imagery time as follows. First, the average decision cycle time is computed from Soar's total CPU time divided by the total number of decision cycles.[19] The average amount of time spent in SVI per imagery operation (i.e. construction and inspection in this domain) is normalized to the hypothesized decision cycle time where normalized time = (average-real-time-in-SVI-per-imagery-operation x 50) / average-decision-cycle-time. Note that each imagery operation encompasses all of the analytical geometry computations (e.g. calculating the rotation direction between a line segment and the COG of a triangle) required for that particular operation (i.e.

[19] A decision cycle is one iteration through Soar's processing loop (see Figure 6-2)

construction and inspection). In this task, the average normalized time spent in SVI for an imagery operation was less than one-half of a millisecond, suggesting that spatial imagery processing is reactive, as it requires less than one percent of a single decision cycle (i.e. 0.5 ms / 50 ms = 0.01).

### 7.2.3 Geometry Problem Assessment

Although the model is not psychologically plausible as we expect humans would "write" the imagined structure to an external diagram, it does demonstrate imagery's computational advantages and added functionality. The quality of the resulting solution is the same whether the agent solves the problem with Soar-SVI and additional task knowledge or Soar+SVI. The architecture enables this behavior while maintaining responsiveness. It achieves these results because spatial imagery processing uses mathematical reasoning specialized for the quantitative spatial representation.

There are tradeoffs, however. The resulting architecture is more complex and no longer includes one amodal representation. As a result, the agent must determine what the appropriate representation is for a given task or subtask. Although we model this problem as a spatial imagery task, individual differences may influence whether one uses visual imagery to solve the problem. For example, in order to find the vertices the agent may employ visual imagery by generating an image for each visual object (i.e. line) and coloring each object the same (e.g. white). It can then identify the vertices with depictive rules marking each pixel with a unique color when there are overlapping pixels in two or more layers. (e.g. if there is a white pixel in Layer 1 and a white pixel at the same location in Layer 2, 3, or 4 then mark a vertex). The tradeoff is that there are more imagery operations involved (generation and transformation of the visual depiction), but the processing is perhaps more general as it does not have knowledge that it is marking a vertex representing the intersection of two lines. That knowledge remains in symbolic structures.

Achieving such a level of generality is a continuous struggle when considering the integration of new representations, as one, amodal representation is more parsimonious.[20] Consider the geometry experiment. For imagery construction, there is a distinct separation between knowledge and architecture. The agent has a finite set of ways to add visual objects to the spatial representation, and as long as it knows how to use them (e.g. the need to "rotate" line C to have it intersect with line A), then it achieves the desired effects without the architecture having to encode any special task knowledge. The architecture does not know that the visual objects it is composing are lines.

However, the inspection process does not provide as clear of separation as the processing immediately exploits the knowledge that the visual objects are lines.[21] The fundamental issue is not whether the architecture should exploit the dimensionality (i.e. one-dimensional line, two-dimensional polygon, etc.) of a visual object, as we assume that at some architectural level there are low-level primitives for processing geometric representations of lines and their relationships. Rather the issue is whether these geometric properties (intersect, parallel, vertices, line-segments) belong to the set of spatial property primitives (direction, distance, orientation, size, and topology) communicated between the symbolic and quantitative spatial representations.

The alternative is to recast geometric types as a combination of other spatial properties. For example, we previously described how constructing two parallel lines can be rephrased as "left-of/right-of" (direction) and disconnected (topology). Similarly, for inspection, geometric intersection may be interpreted as the topological property of "partially overlaps." The underlying architecture must then have mechanisms to determine whether the visual objects are one-, two-, or three-dimensional representations, perhaps by determining the dimensionality of their convex hulls and invoking the appropriate processing. The resulting shape returned to Soar, rather than being labeled a vertex, would simply be labeled a general shape with one point. The agent can then infer, perhaps through semantic knowledge, that a shape with one point represents a vertex.

---

[20]Unless the architecture has to support vision as well in which case reuse of visual perceptual mechanisms, as we demonstrate in this dissertation, may be more parsimonious than mimicking imagery-type operations with the laborious churning of symbolic processing.

[21]The current implementation of the primitive visual objects, or scene graph leaf nodes, labels the nodes as "mesh" (i.e. vertices and indices), "line" (i.e. two vertices and one edge), or "point". Separate processing paths for each type of primitive object are then possible.

These are the types of issues we propose investigating for future work to include the integration between imagery processing and long-term semantic and episodic memories.

## 7.3 Alphabet Soup

In the geometry experiment, we explore spatial imagery and show that for tasks involving more than a few spatial properties, using a symbolic representation and logical reasoning is sufficient but not as computationally efficient as using the quantitative spatial representation and mathematical processing. In this experiment, our desire is to explore tasks where the use of visual imagery and depictive representations is more efficient and likely necessary to infer the desired information.

The domain derives from Thompson et al. (in press). In this experiment, the subject hears a letter from the English alphabet and is instructed to visualize it in its uppercase format. Next, the subject hears a cue, such as "curve," "enclosed space," "horizontal symmetry," or "vertical symmetry" and indicates, as quickly as possible, whether the letter has the particular feature (Figure 7-6). Response times are measured. For example, the letter 'A' has an enclosed space and vertical symmetry while 'U' has a curve. We outline further experimental details in Appendix A.

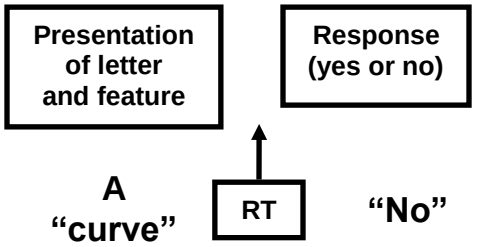



**Figure 7-6: Alphabet Features Experiment**

We chose this task because, unlike the geometry problem, reasoning with symbolic or quantitative spatial representations may not be able to infer the required information without explicitly encoding every feature or task knowledge to support the inference (e.g. if three non-collinear lines then there is an enclosed space). Since an object can have an infinite amount of features, that approach is not scalable from either a psychological or a computational perspective. The task also includes an external environment (i.e. the person asking the question) emphasizing the interaction of imagery and cognition.

Similar to the human experiment, the Soar+SVI agent "hears" a question, visualizes the letter, searches for the desired feature, and then "verbally" responds. The simulation models the audio input as a sequence of two symbols (e.g. "A curve") and the verbal output as a simple "yes"/"no" symbolic response. The agent's initial, declarative knowledge is a symbolic structure containing the 26 capital letter entities. Each letter symbol has the "name" of the letter (e.g. "A") for identifying it with the "audio" input and the visual object symbol for the letter to support imagery. The agent's symbolic knowledge does not have any specific letter features.

After hearing the question and searching its symbolic memory for the desired feature, the agent realizes it cannot infer the answer directly and constructs the quantitative spatial representation from the visual object of the letter. It then generates the visual depiction for the letter, invokes SVI to inspect the Visual Buffer for the desired feature, and "verbally" responds after receiving the results. The simulation records the time from when the agent receives the question to when it provides an answer.

The architecture detects curves using a Hough transform and enclosed spaces using depictive manipulations encoded in the architecture. Appendix B describes these algorithms in more detail. SVI's inspector determines symmetry by first generating a second image of the visual object for the letter and transforming (i.e. rotating) it around the given axis of symmetry. It then marks (with a different color) the pixels on the rotated image that overlap (i.e. have the same color pixel) with the original image. If the number of remaining pixels of the original color is below some threshold, then the inspector considers the letter symmetrical around the given axis of symmetry.

### 7.3.1 Alphabet Results

The requirement for generating and transforming depictive representations to infer features forced us to reconsider our original theory and design that did not include a visual depictive representation. Our previous discussion of the theory and architecture reflects this subjective assessment. A notable observation from this assessment is that there is a difference between the types of algorithms capable of processing a depictive representation. Some are mathematical-based while others take advantage of the color, topological space, and locality of neighboring pixels. That is, even though the

representation is depictive, the processing may be sentential or depictive. Many accounts of visual imagery processing do not clarify this distinction, and, as Anderson (1978) argues, every computational account of imagery must not only discuss the representation but also the types of processing.

As an example, the Hough algorithm maps edge pixels from an image space to a parameter space and uses a "voting" algorithm to determine the parameters of a curve. Although the algorithm has interesting perceptual characteristics (e.g. edge detection), it relies on sentential, algebraic computations. On the other hand, algorithms such as pixel-level rewrites (Furnas, 1990, 1991; Furnas et al., 2000), which we employ for detecting enclosed spaces, are depictive computations as they manipulate the representation based on the spatial configurations and color of neighboring pixels.

The second observation from this experiment is that the gain in computational efficiency, functional capability, and quality of the problem solving is obvious. We cannot achieve this capability or solve these types of reasoning problems, at least in a practical sense, with symbolic or quantitative computations.[22] Therefore, a comparison of computational efficiency between agents with and without imagery is immaterial. As an alternative, we compare the Soar+SVI agent's performance with human data. We make no claim that the algorithms are similar to how humans recognize these features, but we use the data to highlight shortcomings with the architecture.

Figure 7-7 shows the comparison for each feature (horizontal and vertical symmetry are combined). We sort the letters for which we have human data[23] along the x-axis from left to right according to human response time. The y-axis represents the response time in milliseconds. We scale the Soar+SVI agent response times for curves (1/10) and symmetry (2x) to fit within the human bounds. Although there is no correlation between the Soar+SVI agent and the human response times for individual letters, both humans and Soar+SVI show variability in the time to detect curves and enclosed spaces between various letters (Figure 7-7a&b). In the case of symmetry, however, Soar+SVI shows little variability while humans show a lot (Figure 7-7c).

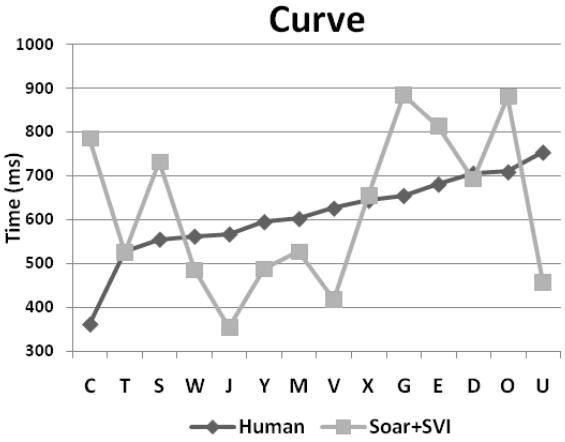
---

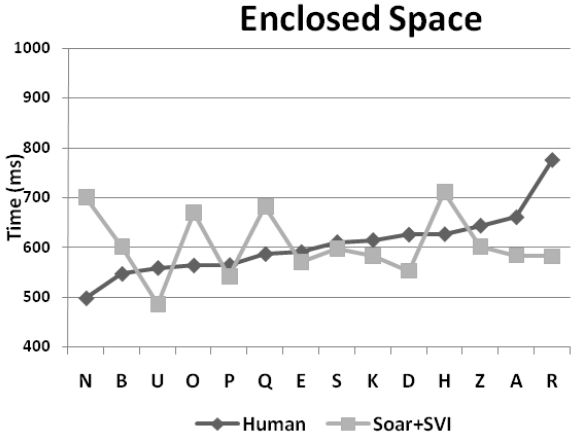
[22] As we explore in the Scout domain, there may be ways to have a symbolic processor, such as Soar, process low-level pixel data. However, as we will demonstrate, it is computationally expensive. Symbolic representations and computations do not facilitate efficient manipulation of depictions.

[23] We would like to thank (Thompson et al., in press) for the data and collaboration on this experiment.

A possible reason why Soar+SVI is consistent in recognizing symmetry while humans are not, is that the architecture does not account for the human phenomena of having to continually "refresh" the visual depictive representation as it fades due to perceptual interference (Kosslyn, Thompson, & Ganis, 2006). Additionally, the agent determines symmetry by transforming the original depiction around the axis of symmetry and comparing it with the original orientation. Rather than performing this operation in a single step, we hypothesize that humans must continuously rotate and regenerate the letter where the time to rotate the object is linear to the rotational angle (Shepard & Metzler, 1971). The results demonstrate that even if the overall architecture is correct (our hypothesis), the details of modeling human behavior is in low-level visual processing.

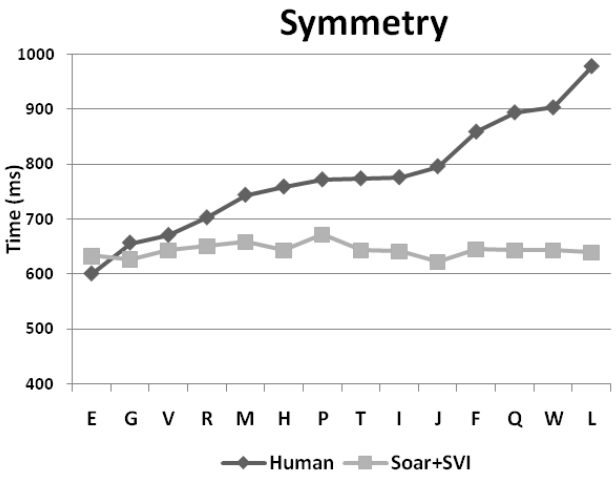
These details become more evident when we measure the responsiveness of the architecture. Figure 7-8 illustrates the average amount of time for one imagery operation (construction, transformation, generation, inspection) normalized to the hypothesized decision cycle time in humans (log 50 ~ 1.7). Most of the reflected time is for inspection of the particular visual feature (i.e. visual imagery). Response times for determining enclosed spaces and curves are respectively one and two orders of a magnitude higher from what we consider supporting responsive behavior (i.e. ~50 ms). There are a couple of reasons for this seeming discrepancy. First, we assume that low-level visual processing exploits parallelism that we do not reflect in our architecture. Parallel hardware and algorithms may help, but a more fundamental issue may be that the architecture is attempting to process too much of the image in a single decision cycle. We attempt to address this issue with the incorporation of an attention window and evaluate it in the Scout domain. Second, it may be that visual imagery simply takes more processing time than spatial imagery using traditional computer hardware. Comparing the normalized response time from the geometry problem (< 1 ms) with Figure 7-8 suggests just that.




**(a)Human** $\mu : 610, \sigma : 98$ **,Soar** $\mu : 611, \sigma : 178$

**(b) Human** $\mu : 604, \sigma : 65$ **, Soar** $\mu : 605, \sigma : 65$




**(c) Human** $\mu : 778, \sigma : 104$ **Soar** $\mu : 643, \sigma : 12$

**Figure 7-7: Comparison of Response Times to Detect Alphabet Features**

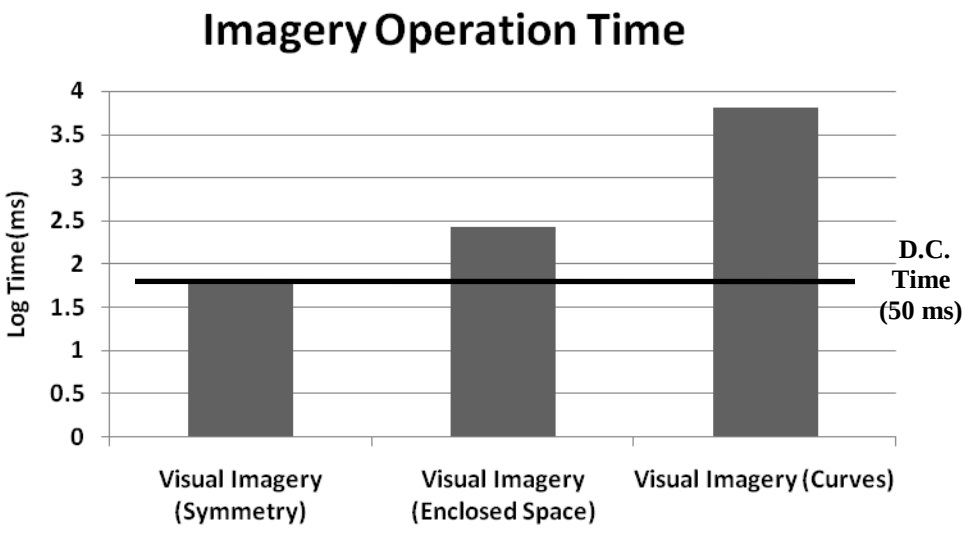



**Figure 7-8: Time Required for an Imagery Operation in the Alphabet Experiment**
**Time is an average of all imagery operations normalized to the decision cycle time (50 ms).**

### 7.3.2 Alphabet Experiment Assessment

The results show that visual imagery provides a functional advantage over symbolic processing when recognizing visual features. However, visual imagery appears to be less reactive then spatial imagery, at least in light of our current implementation. Since perceptual input in this task does not "interrupt" the agent (i.e. the agent has as much time as required to answer the questions), it can imagine the situation for as long as it desires. However, in a task requiring reactive behavior, as in the Scout domain, the architecture must support the interruption of imagery processing when "important" perceptual input is competing for cognitive and visual resources.

Similar to the Geometry experiment, the Alphabet experiment demonstrates a clear separation between knowledge and the architecture for imagery construction and generation. Knowledge directs the processing of these functions and the architecture does not know if it is constructing or generating a letter from the English alphabet, the Chinese alphabet, or another visual object. It only knows that it is activating objects from VLTM based on a given spatial configuration.

Questions linger, however, regarding the knowledge transparency for the inspection processes. As with the geometry problem, our assumption remains that at some architectural level there are primitives for detecting basic features such as lines,

curves, corners, enclosed spaces, etc., as visual processing requires these detectors to automatically perform their role during bottom-up processing. The algorithms and depictive manipulation rules for detecting these features are "hardwired" into the architecture (or acquired early on in development).

However, consider alternative recognition schemes for symmetry. Perhaps it is a property that is not "hardwired." Rather the recognition emerges from the combination of symbolic and perceptual representations where the agent's procedural and declarative knowledge plays more of a role (i.e. algorithms of a more richly hybrid sort). Within the context of Soar+SVI the processing may be realized when an agent's procedural knowledge determines it wants to recognize symmetry and generates two images of the letter transforming the second image around the given axis. Then, rather than having an architecturally embedded algorithm deciding if an object is symmetrical, an imagery command sends depictive rules to SVI. SVI marks the non-overlapping pixels in accordance with the rules and returns the resulting symbolic representation of the shape(s). Based on the number of points in the shape, the agent's procedural knowledge decides whether the object is symmetrical. Not only is this type of recognition for symmetry possibly relevant for imagery, but it is also relevant during perception where the inspection for it initiates after bottom-up processing fails to recognize an object from more primitive features.

From a psychological perspective, the top-down explanation of how one may recognize symmetry clarifies to some extent why humans take longer, on average, to detect symmetry than to detect curves or enclosed spaces (Figure 7-7). Functionally, the advantage of this type of processing is that it facilitates learning as the procedural knowledge of how to recognize symmetry is available to Soar's learning mechanisms. This observation helped inform our decision to encode the knowledge embedded in depictive rules, at least for "non-primitive" manipulations, within Soar's symbolic memories and send them to SVI's Visual Buffer Manipulator for processing. The Scout domain explores this design in more detail.

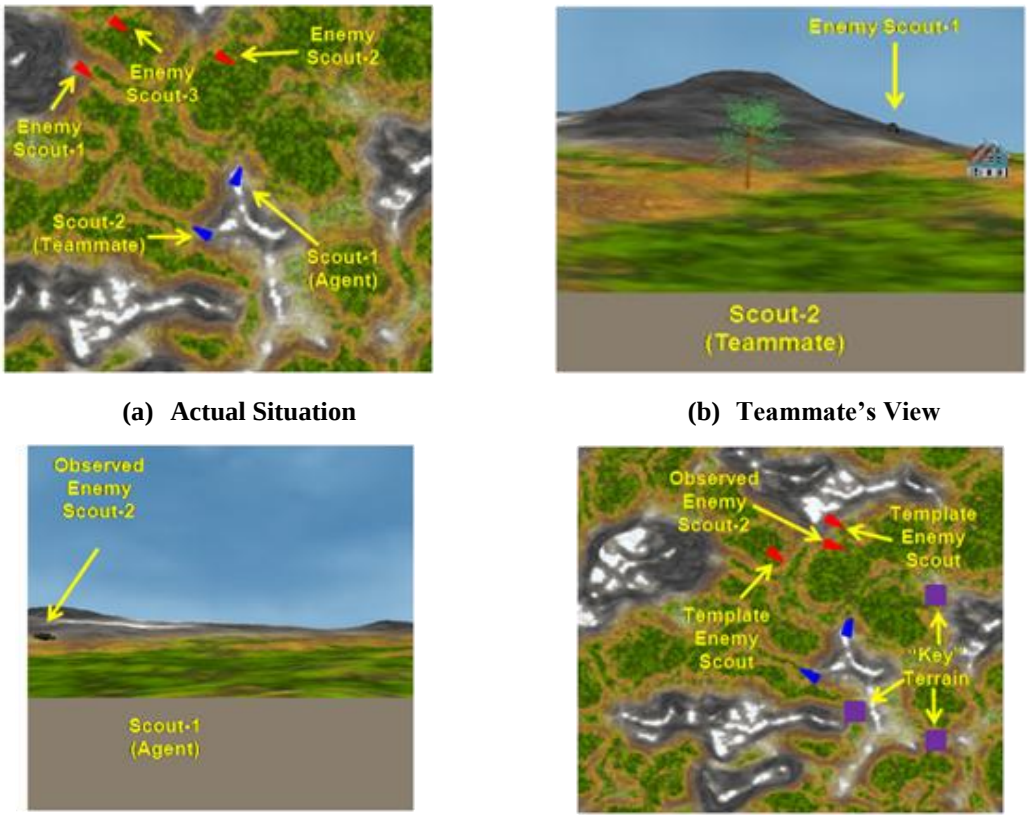
## 7.4 Scouts Out

Our experimental results from the previous two domains are limited to solving internally represented problems. Although the Alphabet experiment includes perceptual input, it is not of the visual modality so the architecture does not have to consider issues such as the difference between perceived and imagined objects and the resource constraints between shared perception and imagery memories. Furthermore, the high-level goal of the previous two tasks was to answer a single question whereas more complex problem solving requires internally answering multiple questions where the information supporting the answer originates from multiple sources—both internal and external.

In the Scout domain, we extend our results to a rich, dynamic environment where perception and imagery operate simultaneously. The agent must interpret and act upon information from multiple sources. By combining perceptual representations with task specific, procedural and declarative knowledge, an imagined situation supports higher-level decisions. Analysis emerges through the manipulation of symbolic, quantitative spatial, and visual depictive representations and provides the agent with the knowledge necessary for reasoning and producing action in the environment.

### 7.4.1 Simulation Environment

To review, there are two scouts in the simulation (Figure 7-9a). One scout is a Soar+SVI/-SVI agent and the team's lead. The other scout, the teammate, is a scripted entity. The opposing force is a scripted, three-vehicle enemy reconnaissance unit that is attempting to determine possible routes and friendly locations in support of a follow-on attack by a larger force (the follow on attack is not modeled). The scout team's goal is to acquire and maintain visual contact with the approaching enemy to determine their maneuver intentions. Paramount in achieving this goal is keeping their commander informed of the opposing force's movements by periodically sending observation reports (through the lead) of their best assessment of the enemy's location. The commander uses this information to position/reposition other friendly forces and reallocate assets (not modeled).




**(a) Actual Situation** **(b) Teammate's View**

**(c) Agent's View** **(d) Agent's Perceived Map / Imagined Situation**

**Figure 7-9: Scout Domain (Scenario-1)**

**The agent's imagined situation in (d) is based on its current perceived/hypothesized knowledge. It has not yet received the teammate's report of the second enemy vehicle (Enemy Scout-1).**

Both the agent and its teammate scan the area in front of them and send reports when they observe enemy vehicles (Figure 7-9b&c). Because of terrain occlusions, the agent cannot observe its teammate but instead maintains situational awareness through the teammate's periodic messages that identify its position. Both the agent and the teammate can reorient their views, but the teammate performs this action only when the lead directs it. The agent can look at the scene or its map (Figure 7-9c&d), but not both simultaneously. We assume that the agent and its teammate can distinguish enemy vehicles from other objects. However, the agent has to decide whether a sighted or reported enemy is a new or previously identified entity.

The simulation architecture consists of five major components (Figure 7-10): a World Representation, a Simulation Engine, several Simulation Actors, an Agent Proxy,

and Soar+SVI.[24] We will discuss each of these components to provide an understanding of the modeling fidelity between the agent and the environment. The World Representation is a Wild Magic (Eberly, 2005) scene graph and includes the terrain and each object in the world (e.g. tanks, buildings, etc). Its purpose is twofold. First, from a simulation perspective it maintains the location and orientation state of each actor and provides a graphical representation of the simulation for debugging purposes. Second, the Agent Proxy provides the agent, as perceptual input, the portion of the world that is visible to the agent. We will discuss this implementation detail shortly.

A Simulation Actor represents a unique entity in the simulation such as the agent, the teammate, the enemy, and buildings. Each simulation actor includes a set of basic behaviors (i.e. move, turn, send a message, etc.) and maintains a reference to its scene graph object representing its 3D model, where it is located, and its orientation. During initialization, the simulation creates the world and the actors from a configuration file. The file contains actor attributes such as their name, 3D model, initial location and orientation, movement velocity and angular velocity. For enemy actors, the simulation also loads the enemy "plan" to include a set of waypoints, possible paths between waypoints, and a few "decision points" where the enemy commander has the latitude (based on randomness) to split its force or modify the path of a subordinate. A tactical expert creates the enemy plan from an off-line analysis.

The Simulation Engine is a discrete event simulation (DES) using the SimKit framework (Buss, 2002). At the most basic level, a discrete event simulation consists of a clock, an event list, and set of possible events that the system schedules for execution by placing them on the event list. During each "tick" ($t$) of the clock, the simulation engine removes all events on the event list whose scheduled execution time is less than or equal to the current simulation time and processes their actions. After the simulation processes those actions, it increments its clock and the processing iterates. The general algorithm is the following:

[24]During the discussion of the simulation when we refer to Soar+SVI, we also imply Soar-SVI as having he same external interface with the simulation.

1. Create and initialize actors and world
2. Set simulation time to 0
3. Schedule the default ("Run") event
4. While there are events on the event list
    a. While there are events on the event list with an execution time <= simulation time
        i. Execute the next event
    b. Increment the simulation time

The actions of an event change the state of the system and may schedule another event for future processing. From the perspective of the simulation, then any action that occurs in the world is the result of a scheduled event. The advantage of this approach is that the system can enforce temporal constraints on actions.

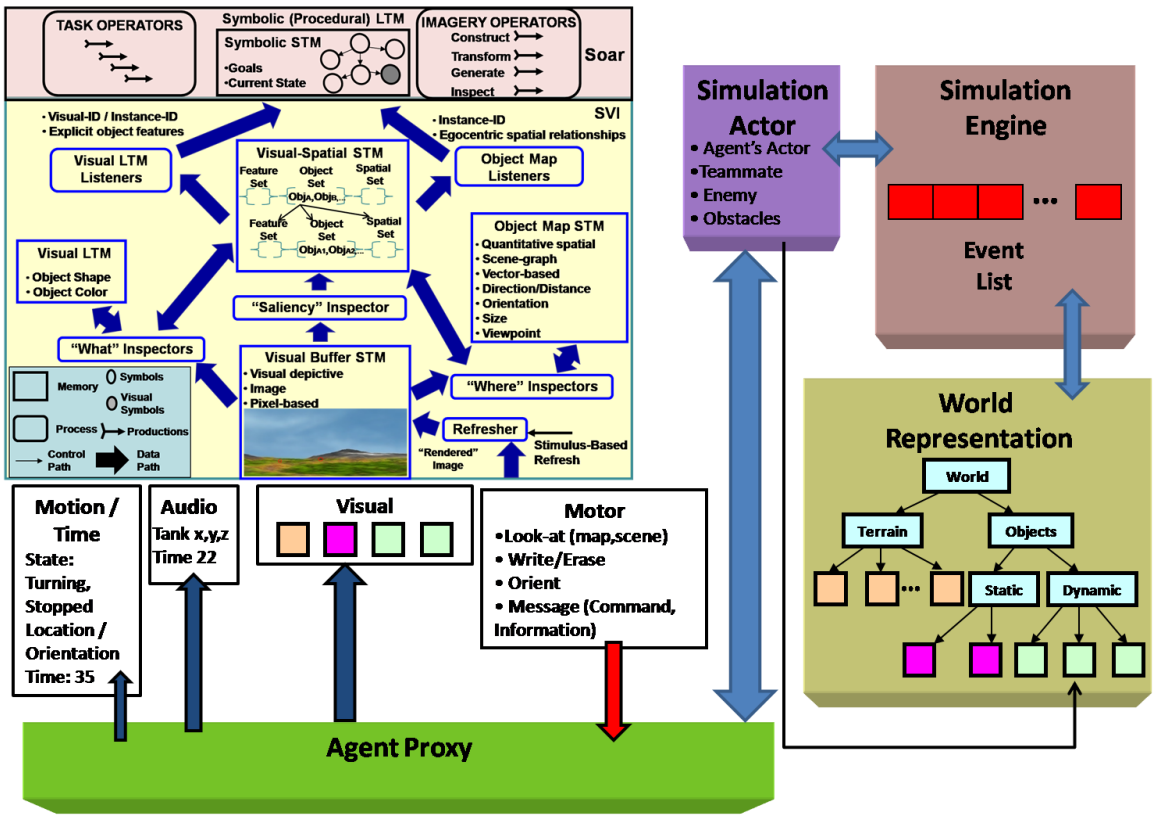



**Figure 7-10: Simulation Architecture**

For example, the simulation models the initiation of a tactical maneuver when one actor sends a message to another actor asking it to commence movement (e.g. reorient, start-move). Each actor can send a message to another actor by scheduling a "send

message" event. The temporal delay between when the actor schedules an event and when it executes can be immediate (i.e. $t + 0$) or sometime in the future, $t + \Delta$, where $\Delta$ is a constant or generated random number based on a given statistical distribution. When the "send message" event executes, the simulation changes the state of the message from "waiting" to "in transit" and schedules a "receive message" event. At some point in the future, the simulation engine processes the "receive message" event by marking it as "received" and delivering it to the appropriate actor for further processing. The actor "reading" the message may, based on its information, schedule another event. For example, when the teammate receives a message from the agent to re-orient, the teammate schedules a "turn" event. Likewise, when an enemy actor receives a message from the enemy commander to move to a checkpoint, the subordinate enemy actor schedules a "move" event changing its status from stationary to moving. Note that this state change does not reorient or move the actor's scene graph object. The dynamic change occurs through another event that we explain next.

To model animation, the simulation schedules a special "update world" event that, after processing, reschedules itself for the next simulation tick so that it continuously processes (Buss & Sánchez, 2005). The simulation schedules this event with the lowest priority so that it executes only after all other events scheduled for that time have an opportunity to run. The basic algorithm for the update world event is the following:

1. For each actor
    a. If motion status is turning or moving, update scene graph object's location and orientation based on the actor's movement/angular velocity (Newtonian physics)
    b. If one or more "opponents" are visible and actor has not sent a report in "awhile", then schedule event to send message to leader (NOTE: does NOT apply for the agent actor).
    c. Record status (for logging / debugging)
2. Referee
    a. Determine what objects each actor can observe (*detection algorithm*)
    b. Tell each actor what they can observe
3. Notify Simulation Clients
    a. Simulation user interface (for rendering the scene)
    b. Agent Proxy (to provide perceptual input to Soar+SVI)
4. Schedule another update world event with the lowest priority for time $t + 1$

The update world event gives each simulation actor an opportunity to update its dynamic state, such as location and orientation and report status changes if it has not done so in awhile.[25] The event processing then referees the current situation by determining what objects are visible to each actor using the following detection algorithm:

1. For each actor that can "see" (i.e. not buildings)
    a. Determine the objects in the actor's view frustum and sort closest to furthest
    b. For each object in view frustum
        i. If on periphery (camera plane), remove object
        ii. If a closer object in view occludes it (ray tracing), then remove object
        iii. If terrain occludes object (ray tracing), remove it
    c. Give actor its list of remaining visible objects.

The occlusion detector uses a ray-tracing algorithm and checks three different points on the object (top and two sides). The detection algorithm assumes the object is visible if two of the three points are visible (i.e. intersects the ray). Finally, the update world event notifies clients, such as the simulation's user interface and the Agent Proxy that the event is complete so they can process. This step is where the Soar+SVI agent receives its perceptual input from the Agent Proxy.

The Agent Proxy serves as an interface between Soar+SVI and the simulation (Figure 7-10). After the update world event notifies the Agent Proxy that it can process perceptions, the proxy sends the agent its current perceptions through SVI. The perceptions include audio (i.e. messages), motion status, simulation time, and visual. For this domain, SVI has components to process the audio, motion, and time input from the Agent Proxy. These components simply pass the input on to Soar by creating symbolic structures on Soar's input-link.

The visual input is a copy of the individual scene graph nodes from the World Representation that the detection algorithm determines the agent can observe. The Agent Proxy provides this list of nodes to SVI's Saliency Inspector. The first node in this list is always the "background" node, which includes the terrain. As discussed in Chapter 6, the Saliency Inspector instantiates the structure of the current scene in VS-STM and provides

[25]Defined as the opposing entity moving more than 500 meters from last reported location or 60 simulation ticks.

the "What" and "Where" Inspectors this list of scene graph nodes. The "What Inspectors" recognize the objects with a simple "string match" and the "Where Inspectors" build the internal Object Map scene graph from the provided list of nodes. For debugging purposes, the Refresher renders the scene graph from the agent's current viewpoint. The remaining processing proceeds as discussed in the architectural description.

Although we do not literally model the two "what" and "where" control paths (see Figure 6-7), we do model the separation of control between cognition and perception with two separate threads. The Agent Proxy's thread processes the perceptual input, up to the point where SVI creates the structures in the Object Map and VS-STM (i.e. the "what" and "where" processing). Simultaneously, another processing thread is executing Soar's decision cycle beginning with input received from the VLTM and Object Map Listeners.[26]

Recall that the visual perceptual processing in SVI automatically creates symbolic structures for Soar to include the recognized objects in the scene and their direction, distance, and relative size from the agent's egocentric perspective. In order to determine the object's orientation in this domain, the agent has to observe the entity move and then infer its orientation, or direction of travel, by taking into account its current and previous locations. One important point is that when the agent is looking at the scene, the scene graph nodes represent objects in the environment. When the agent is looking at the map, the terrain nodes represent the map and the salient, visual objects are the map-icons that the agent has previously "written" on the map. In this case, rather than providing Soar the egocentric direction and distance of the map icons from the agent's eyes and then forcing it to "read" the map to infer absolute locations, we simply provide the absolute locations and orientations. The agent's procedural knowledge must interpret the incoming direction and distance accordingly, based on whether it is looking at the scene or the map.

When Soar sends commands to SVI, they may be imagery commands or instructions to create action in the environment. As with the extra perceptual components (i.e. audio, motion) in this domain, SVI also has some motor components that receive commands from Soar and communicate the instructions to the Agent Proxy. These motor commands include changing the agent's view from looking at the scene to looking at the

[26]Unless running on multi-core machine, the threads are interleaved.

map (or vice versa), writing an object on the map, orienting in a given direction, or sending a message. The Agent Proxy interprets these commands and schedules the events through the agent's Simulation Actor. When the simulation processes the event, it produces the corresponding action in the environment by updating the internal state of the agent's Simulation Actor. The agent perceives the action's results via the Agent Proxy.

Again, note that these events take time to process based on the modeling of their temporal constraints. Therefore, when the agent issues a command to write an object on the map or look at the scene, Soar will execute a few decision cycles before it perceives the change based on the action. This modeling produces some interesting behavior. For example, to write an object on the map the agent must first imagine it by constructing the quantitative spatial representation of the map icon. The agent can then issue a write command to simulate writing the icon on the external map, in effect making it persist. However, while writing the map-icon, the agent may be interrupted (e.g. the teammate sends a message), forcing it to attend to the perceptual input. Because "important" perceptual processing inhibits imagery, the imagined map-icon "disappears" before the agent has had an opportunity to finish writing it. The agent has to re-imagine and rewrite the object. The architecture's truth maintenance mechanisms must consider these types of perception and imagery interactions, for example by removing the imagined visual-object structure of the map-icon from Soar's short-term memory and removing any stale state in SVI referring to its visual-object. The agent's procedural knowledge must also wait until its symbolic perceptions inform it that it is perceiving, rather than imagining, the map-icon and not assume that just because it sent the command the action occurred.

### 7.4.2 Task Decomposition

The primary goals of a scout team are to acquire and maintain visual contact with the approaching enemy and report all information rapidly so that the commander can make his assessment. After establishing visual contact, a scout's actions include the following steps (Army, 2002).

(1) Deploy and report
(2) Analyze the situation
(3) Choose and execute a course of action

Analyzing the situation involves reasoning about known friendly and enemy locations and orientations, terrain, and obstacles. If the scout lead does not know the locations of all expected enemy, then he might hypothesize the location of other enemy entities and template their positions. Based on the analysis, the scout lead then decides if he should reorient himself, his teammate, or both.

Figure 7-11 illustrates the Soar+SVI agent's task decomposition to perform this mission. The left-to-right ordering implies some task sequence although this is not a hard rule, as the reasoning is conditional, iterative, and sometimes interleaved. Tasks in bold, italic font use imagery operations to assist in the reasoning. Figure 7-11a shows the top-level tasks. During a reconnaissance mission, the scout agent is continuously observing the situation, analyzing its possible courses of action, and deciding and acting on a course of action. Observing (Figure 7-11b) includes scanning the area attempting to establish visual contact with the approaching enemy. Once the agent or its teammate makes contact, the agent must identify whether the entity it is observing, or the teammate is reporting, is a new enemy or a previously identified enemy. We assume that the agent can distinguish between two different entities that it observes. For enemy vehicles that the teammate reports, however, the agent uses the quantitative spatial representation to determine if the distance between the reported location and a previously identified enemy is within a certain threshold (e.g. 150 meters). If the constraint is true, the agent assumes that the teammate is observing the same entity the agent is observing. After identifying the entity, the agent records the observation and sends a report to the commander. If the agent is confident in the observation it writes it on the map by first imagining the map icon (construct) and then issuing a "write" command.

Analyzing the situation (Figure 7-11c) begins by first determining if the agent has an observation for all expected enemy (i.e. three), and, if not, hypothesizing (i.e. template) the unknown enemy locations. The hypothesis involves retrieving semantic knowledge of a typical enemy reconnaissance element (e.g. three vehicles, one vehicle forward, two vehicles behind at a given distance and orientation) and imagining their locations (Figure 7-9d). The agent may also employ its knowledge of a typical enemy

vehicle's velocity to simulate movement when the last known location of an enemy is stale (i.e. recorded time is less than the current time).

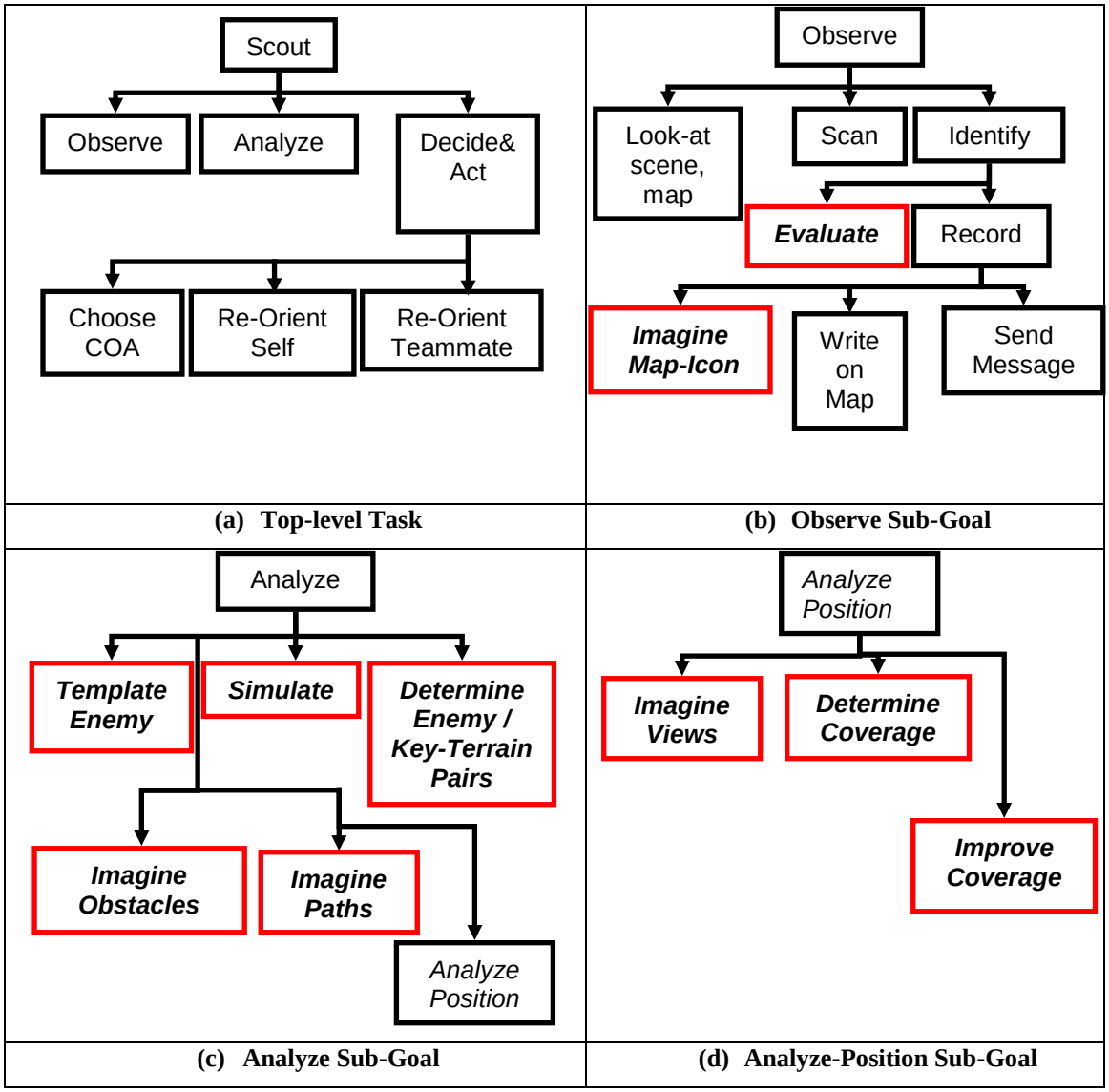


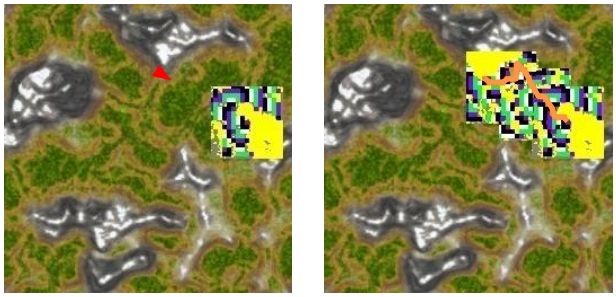

**Figure 7-11: Scout Domain Task Decomposition**

To analyze each enemy's avenue of approach or path, the agent must first make an assumption as to their next destination. An enemy can be terrain-oriented or force-oriented. For this scenario, the agent assumes the enemy is terrain-oriented. One way the agent may determine an individual enemy's next destination is for each piece of

hypothesized key terrain,[27] use spatial imagery to infer the direction, orientation, and distance from the enemy vehicle to the key terrain. With this information, the agent may employ a heuristic, such as preferring the minimum distance and change in orientation, which forms a hypothesis as to the individual enemy's next destination. The agent repeats this reasoning for each known or hypothesized enemy.

After hypothesizing each enemy's destination, the agent uses visual imagery to imagine known obstacle barriers and restricted or "slow-go" terrain (e.g. vehicles cannot drive through buildings, vehicles have problems with steep terrain, enemy scouts will attempt to avoid open spaces, etc.). It then marks a path by transforming the visual depictive representation. The agent's knowledge provides imagery with the enemy vehicle (source) and key terrain (sink) visual objects, the attention window parameters, and the set of rules for the depictive manipulations. Imagery then performs the manipulations, marking the path. The general procedure is the following (Appendix B provides more details):

1. Mark all known obstacles and "slow-go" terrain with a color (yellow) by applying a set of threshold values.[28] Mark all other pixels gray.
2. Grow an iso-distance contour field avoiding any previously marked barriers (Figure 7-12a).
3. Walk the contour field from source to sink, marking the path along the way. (Figure 7-12b).



**(a) Distance field flood** **(b) Mark Path**

**Figure 7-12: Imagining an Enemy Path**

Finally, the agent analyzes its position (Figure 7-11d) by imagining its teammate's view and its own view by first constructing and generating the views.

[27]Key terrain is any location where control by either friendly or enemy forces offers a significant advantage because it provides good observation of converging paths, is a logistical hub, has psychological implications, etc.

[28]Threshold values are determined from an off-line analysis and encoded as part of the agent's declarative, task knowledge.

Generation takes into account some terrain occlusions (e.g. that a hill blocks the view). The agent determines the amount of coverage by inspecting the Visual Buffer images for the size (i.e. length) of each hypothesized path that is a topological proper part of each view (Figure 7-13). Soar's procedural knowledge then estimates the amount of coverage a particular view has on a path by dividing the covered portion of the path by the total path length. If there are paths with coverage below a certain threshold (i.e. 0.25), the agent attempts to improve coverage by simulating reorientations of its teammate, itself, or both. Again, the agent uses a combination of imagery and task heuristics embedded in procedural knowledge to determine who to reorient, what direction to reorient them, and how far to simulate the orientation. To execute the simulation, the agent issues imagery commands to transform and regenerate the views. It then re-inspects the visual depictive representations in the Visual Buffer to determine the resulting coverage. Based on its analysis, the agent makes a decision (Figure 7-11a) and issues a motor command to reorient itself and/or send a message to its teammate directing it to re-orient. The agent starts observing the scene again and the "observe, analyze, decide, and act" process continues.

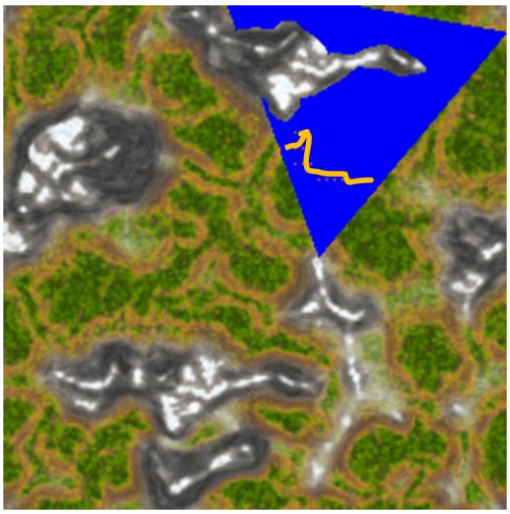


**Figure 7-13: Agent Imagining Its Coverage of One Enemy's Hypothesized Path**
**Agent also imagines coverage for other paths and its teammate's coverage of each path**

As a summary of the agent's analysis and subsequent decision, Figure 7-14 shows a trace from one of the runs. Bold font indicates our own remarks to clarify the trace. Note that the agent's reasoning is non-trivial and that its functionality and problem-

solving quality emerges from the combination of symbolic, quantitative spatial, and visual depictive representations.

**Results from the analysis of Scout-1's (Agent) and Scout-2's (Teammate) current path coverage.**
**cf = coverage factor (length of path covered by view / total length of path)**
**distance = distance from entity's location to path center in meters**
**orientation distance = how far entity has to orient (in degrees) to be in line with path center**

Scout-1's path analysis path: P49 cf: 0. distance: 1277.63 orientation distance: 49.4096
Scout-1's path analysis path: P51 cf: 0.0663824 distance: 590.454 orientation distance: 53.0644
Scout-1's path analysis path: P53 cf: 0.460498 distance: 851.323 orientation distance: 3.64507

Scout-2's path analysis path: P49 cf: 0.676294 distance: 1373.78 orientation distance: 19.3111
Scout-2's path analysis path: P51 cf: 0.126035 distance: 772.466 orientation distance: 38.1506
Scout-2's path analysis path: P53 cf: 0. distance: 1337.91 orientation distance: 63.6824

**Agent decides to simulate reorienting itself (Scout-1) to cover path P51. The reason it chose P51 over P49 is because Scout-2 already has adequate coverage on path P49**

Reinspecting path-coverage by attempting to improve Scout-1's view to cover path P51

**Analysis results after simulating the reorientation and inspecting for path coverage.**

Scout-1's path analysis path: P49 cf: 0.591979 distance: 1277.63 orientation distance: 3.5553
Scout-1's path analysis path: P51 cf: 0.340073 distance: 590.454 orientation distance: 0.0989148
Scout-1's path analysis path: P53 cf: 0. distance: 851.323 orientation distance: 56.6101

**Agent sees that after simulating the reorientation it has better coverage of path P51. However, it decides to reject the COA because its takes it off its primary path, P53. That is, its coverage factor of path, P53 has dropped below 0.25 and in the previous, initial analysis, the agent's orientation distance to path P53 is 3.6 degrees (i.e. it has to orient 3.6 degrees to be oriented exactly on the path's center) whereas Scout-2's orientation distance to P53 is 63.7 degrees. This is an indicator that Scout-1 currently has primary coverage on path P53. Note, that rejecting this COA does not reject the possibility of reorienting both the agent and the teammate where each would assume responsibility for a new primary path.**

Rejecting COA because takes Scout-1 off its primary path: P53

**Agent decides to see if it can obtain better coverage on path P51 by reorienting its teammate.**

Reinspecting path-coverage by attempting to improve Scout-2's view to cover path P51

**Analysis results after simulating the reorientation of its teammate and subsequent inspection of path coverage. Teammate can adequately cover path P51 while also improving coverage on its current primary path, P49.**

Scout-2's path analysis path: P49 cf: 0.730275 distance: 1373.78 orientation distance: 18.8389
Scout-2's path analysis path: P51 cf: 0.313793 distance: 772.466 orientation distance: 0.
Scout-2's path analysis path: P53 cf: 0.182153 distance: 1337.91 orientation distance: 25.5324

**At this point, the agent has four possible COA's (one of which it has already rejected):**

**COA 1: Do nothing (NOT SELECTED)**
**COA 2: Reorient self (REJECTED)**
**COA 3: Reorient both (NOT SELECTED)**
**COA 4: Reorient teammate (SELECTED)**

**Agent decides to reorient teammate to achieve better coverage on path P51. Each COA along with its advantages follows. Note that even though COA 3 has a better cumulative total coverage, the agent does not select it as it leaves path P53 with less than 25 percent coverage. Coverage factors shown here represent a cumulative total for each path. Possible COA advantages are:**

**better-path-coverage: overall (total) path coverage is better than current**
**maximum-coverage-on-different-paths: Agent's and teammate's maximum coverage are on different paths (i.e. they are not focusing on the same path).**
**all-paths-<twenty-five or fifty or seventy-five>-percent-covered: each path has at least the corresponding percentage of coverage**

**COA 1: Do nothing (NOT SELECTED)**
Non-selected COA coverage factor: 0.676294 for path P49
Non-selected COA coverage factor: 0.192417 for path P51
Non-selected COA coverage factor: 0.460498 for path P53
Non-selected COA total coverage factor: 1.329209
Non-selected COA advantage: maximum-coverage-on-different-paths

**COA 2: Reorient self (REJECTED)**
Non-selected COA coverage factor: 1.26827 for path P49
Non-selected COA coverage factor: 0.466108 for path P51
Non-selected COA coverage factor: 0. for path P53
Non-selected COA total coverage factor: 1.734378
Non-selected COA advantage: better-path-coverage

**COA 3: Reorient Both (NOT SELECTED)**
Non-selected COA coverage factor: 1.32225 for path P49
Non-selected COA coverage factor: 0.653867 for path P51
Non-selected COA coverage factor: 0.182153 for path P53
Non-selected COA total coverage factor: 2.15827
Non-selected COA advantage: better-path-coverage

**COA 4: Reorient Teammate (SELECTED)**
Selected COA coverage factor: 0.730275 for path P49
Selected COA coverage factor: 0.380175 for path P51
Selected COA coverage factor: 0.642651 for path P53
Selected COA total coverage factor: 1.753101
Selected COA advantage: all-paths-twenty-five-percent-covered
Selected COA advantage: maximum-coverage-on-different-paths
Selected COA advantage: better-path-coverage
COA chosen re-orienting teammate to 101.229 degrees

**Figure 7-14: Execution Trace of Scout Agent's Analysis and Subsequent Decision**

### 7.4.3 Computational Advantage

One of our computational constraints is for a component to process its representation efficiently, yet remain task independent. As we discovered from the Geometry and Alphabet experiments, achieving this balance is not always clear. We had to consider similar tradeoffs in the Scout domain as we expanded the architecture to include transformations of the visual depictive representation using pixel-level rewrites. The new component, the VBManipulator, proves useful in determining the enemy's hypothesized paths as the visual depictive representation takes into account the specific shape of the terrain and obstacles. Alternative approaches (shown in the next section) that do not consider the specific shape have a lower problem-solving quality, as they do not provide as much accuracy.

In order to incorporate the depictive manipulations, we had to consider not only the efficiency of the processing to insure it gave the resulting architecture a computational advantage, but also the demarcation between the knowledge and architecture. The architectural description in Chapter 6 (specifically 6.2.2.2.4) and the more detailed description of the depictive manipulations in Appendix B describe the tradeoffs we considered in achieving efficiency (e.g., minimization of the image size through an attention window). In our analysis of the separation between knowledge and architecture, we considered and compared three alternatives (Figure 7-15).

Recall that a pixel-level rewrite system includes a shared image, a set of depictive rules, and processes to interpret the rules and manipulate the image. Clearly, the image structure and processing to modify the image belong in SVI's architecture, but where do the rules, which contain both procedural (i.e. topological structure of each manipulation) and declarative knowledge (i.e. color of pixels to match) reside? One alternative is to create a task specific component including both the procedural and task knowledge (obstacles, slow-go terrain, distance field colors, etc.) in the architecture. This approach is computationally efficient and serves as a baseline measure but if the task changes the architecture must change. A second design choice maintains all knowledge and the majority of processing in Soar where the pixel-rewrite rules are encoded as Soar productions. Each decision cycle SVI's VBManipulator sends Soar the current set of pixel values. The pixel rewrite rules in Soar determine the current match set and send the

results to the VBManipulator for updating the image. An intermediate solution stores the depictive manipulations (i.e. the pixel-level rewrite rules) in Soar. Soar transmits the appropriate task-specific manipulations to the VBManipulator for processing.

Figure 7-15 illustrates the amount of processing time (log scale) required by each of the alternatives where each implementation executes a distance field flood on a 16x16 pixel image. As shown on the left of Figure 7-15, the first, task specific/representation specific implementation is very efficient compared to our other alternatives. Maintaining the majority of computations within the symbolic processor, as shown on the right of Figure 7-15, is two-three orders of a magnitude slower than the alternatives. The reason, similar to the issue with angles in the Geometry problem, is that there are too many alternatives for Soar's rule matcher to consider efficiently when the fringe layer in the distance field has several matches. Depictive processing that iterates over the image handles these fringe layers in a much more efficient manner as it is only considering the local (e.g. 3x3) topological structure at any one instance. The intermediate approach provides efficient execution (middle of Figure 7-15) coupled with flexibility to change task-specific manipulations dynamically. It also provides high-level control so that if "important" perceptual input interrupts imagery processing, the architecture can respond accordingly.

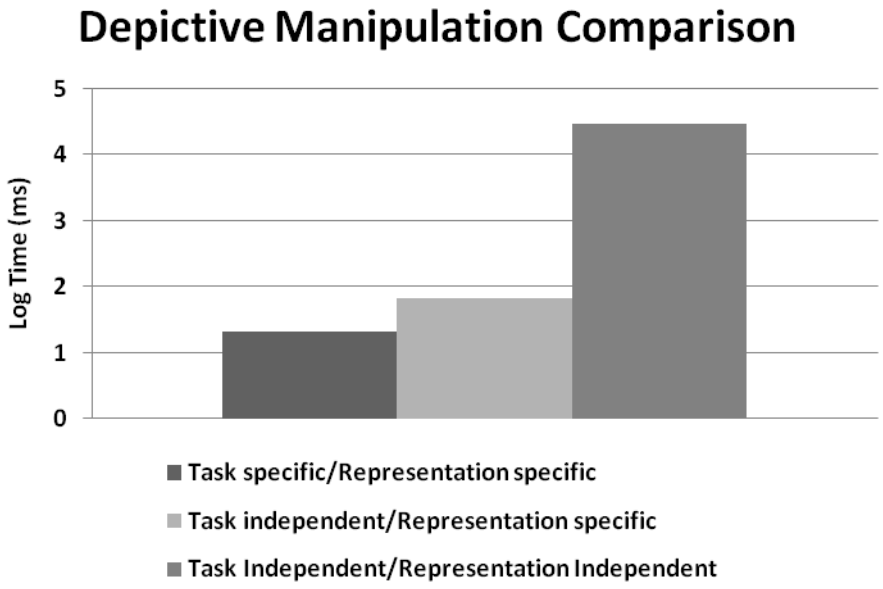



**Figure 7-15: Comparison of Processing Times of a 16x16 Pixel Image**

A final alternative, evaluated subjectively, involves encoding the declarative knowledge in Soar (i.e. the color of the pixels to match), while keeping the procedural knowledge of the rules (i.e. rules for creating distance field flood, rules for marking path) embedded in the architecture. Again, as with the first task specific/representation specific design, we expect to have slightly better efficiency compared with the intermediate solution. However, this requires either knowing a priori every required manipulation that we desire the architecture to process or changing the architecture for every task--defeating the purpose of having a general, purpose architecture. This design choice is more appropriate for manipulations that support the detection of low-level, primitive visual features (e.g. enclosed spaces).

To evaluate the responsiveness of the architecture, as in the Geometry and Alphabet experiments, we compare the hypothesized human decision cycle time of 50 milliseconds to the normalized amount of time for each imagery operation. To review, the average time of a single imagery operation is calculated as follows. First, the average decision cycle time is computed from Soar's total CPU time divided by the total number of decision cycles. The average amount of time spent in SVI per imagery operation (Construct, Transform Object Map, Transform Visual Buffer, Inspect) is normalized to the hypothesized decision cycle time where normalized time = (average-real-time-in-SVI-per-imagery-operation x 50) / average-decision-cycle-time.

Figure 7-16 illustrates the response times ordered left to right with the most responsive operation (Inspect) on the left and the least responsive (Transform Visual Buffer) on the right. The line represents the hypothesized human decision cycle time (log 50 ~ 1.7). Spatial imagery operations (Construct and Transform Object Map) are 2-3 orders of a magnitude faster, and appear more responsive, than visual imagery operations (Generate and Transform Visual Buffer). This result is consistent with our results from the Geometry and Alphabet experiments. The inspection processing includes inspections of both the quantitative spatial and visual depictive representations. Since in this domain, these inspection processes involved relatively few calculations (e.g. direction, distance, and orientation between object; size of path), it was very responsive.

As we previously articulated, the higher response times for visual imagery operations may partially be due to our assumption that low-level visual processing

exploits parallelism, possibly requiring special hardware that we do not reflect in the architecture. For example, a reason for the reflected time to generate a visual depictive representation (~4 ms real CPU time) is that the operation includes the time to copy the rasterized image pixels from the graphical processing unit's memory to main memory. We assume that specialized hardware support would achieve a significant speed up. Even though the manipulation of the visual depictive representation (Transform Visual Buffer) is roughly an order of three magnitudes more efficient when compared to processing the representation with symbolic computations (Figure 7-15), it is not as responsive to the environment as we would like (right column of Figure 7-16). Note that one visual imagery operation (Transform Visual Buffer) includes the execution of many pixel rewrite rules (e.g. the rules to create the distance field flood in one attention window is counted as one imagery operation as its execution is within one Soar decision cycle).

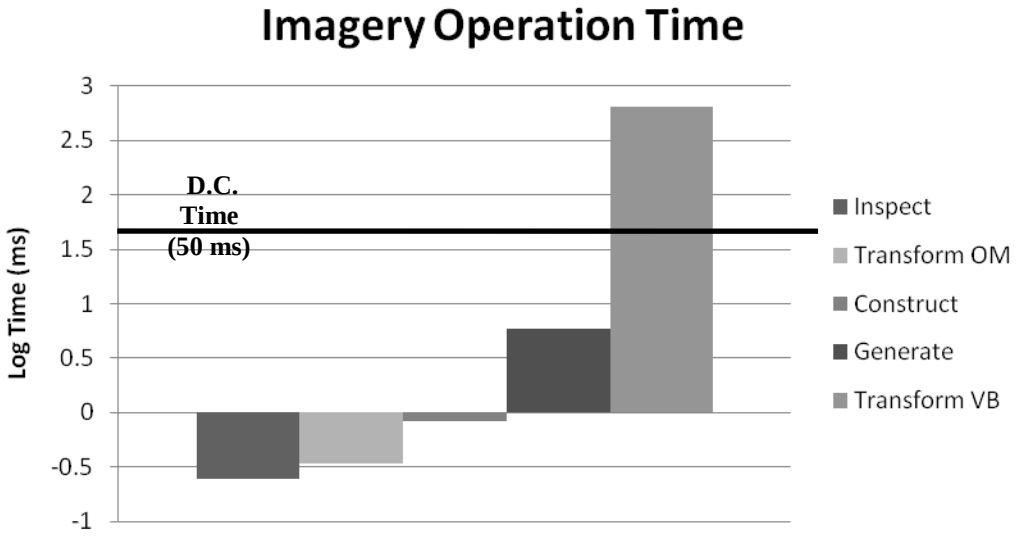



**Figure 7-16: Time Required for an Imagery Operation in the Scout Domain**
**Time reflects an average of an imagery operation normalized to the decision cycle time (50 ms).**

As with detecting curves and enclosed spaces, it may be that the system is attempting to process too much in one decision cycle. Limiting the processing to a 64x64-attention window did achieve significant speedup (~250 ms real CPU time) and appeared "adequate" for this domain. There are also much more efficient pixel-rewrite algorithms (Furnas & Qu, 2002), but their implementation is rather complex and not the main thrust of this research. Increasing the efficiency of the depictive manipulations would only strengthen the claims of this research. From a psychological perspective, it

may be simply that visual imagery take longer to process, suggesting that given a choice, humans tend to use spatial imagery in situations where speed is more important than accuracy.

### 7.4.4 Functional Capability and Problem-Solving Quality

Although reasoning using visual depictive manipulations is not as responsive when compared to the other imagery processes, its advantage is that it provides the architecture with functional capability that the system cannot achieve without it. These manipulations enable the architecture to support reasoning when there are arbitrary shapes involved (e.g. finding paths through undulating terrain, determining the intersection between the view and the winding path). Spatial imagery, using quantitative spatial representations, would require converting the shapes to convex polygons resulting in a loss of accuracy. Better accuracy results in better problem-solving quality.

To evaluate problem-solving quality, we created three agents modeling the lead scout. The first agent (Soar+SVI) uses spatial and visual imagery to observe, analyze, and decide on a course of action. The second agent (Soar-SVI) uses the same task decomposition (Figure 7-11) but uses strictly symbolic and quantitative representations and processing in Soar. For example, it imagines the enemy's paths as straight lines and the views as a triangle without taking terrain or known obstacles into account. Path coverage is determined using basic trigonometry. This is a fair comparison because as just discussed, providing the Soar-SVI agent with all known obstacles to include the no-go terrain would slow its processing significantly (see right hand column Figure 7-15), causing it to miss observations. The third agent (Observer) and its teammate simply observe the area to their front and send reports to their commander without any re-positioning (i.e. it only executes the Observe sub task in Figure 7-11).

To evaluate the generality of the system, we evaluate the same agents in two different scenarios. Figure 7-9 shows the first scenario. We will describe the second scenario shortly. In general, Enemy Scout-1 and Enemy Scout-2 maneuver in the general direction as shown in their initial configuration, but there is some variability in the paths they choose. For example, in Figure 7-9 Enemy Scout-2 may initially maneuver in a southeasterly direction and then make a sweeping maneuver to the south or west. Enemy

Scout-3 randomly decides to follow Enemy Scout-1 or Enemy Scout-2, typically at a distance ranging from 500 to 1000 meters. There are times, however, when it closes that gap to 50-100 meters.

We have two evaluation metrics for problem-solving quality. The first is the cumulative amount of information the commander receives on the enemy's location over time (Figure 7-17). The second metric is the number of reported observations of each enemy entity (Figure 7-18). Both results reflect the average over 30 trials. In Figure 7-17, the x-axis is the current simulation time, and the y-axis measures the amount of information per unit time with 1.0 being perfect information and –1.0 indicating no information. The measure of information is an average over all three enemy entities at simulation time, t, calculated for each enemy as follows:

$$I_t = \begin{cases} -1 \; if \; no \; observation \\ 1 - \delta \; otherwise \end{cases}$$

where:

$$\delta = \sqrt{(obs_x - act_x)^2 + \left(obs_y - act_y\right)^2} \Big/ d_{acceptable}$$

$(obs_x, obs_y)$ is the reported location of an entity at time, t and

$(act_x, act_y)$ is the actual location of an entity at time, t

$$d_{acceptable} = the \; acceptable \; square \; distance = \sqrt{d_x^2 + d_x^2}$$

$$where \; d_x = d_y = 500 \; meters$$

The agent receives a positive score for a given enemy if at simulation time, *t,* the commander's knowledge of the particular enemy's location is within a 500 x 500 meter square of the enemy's actual location at that time. Otherwise, the information score is negative for that time with a minimum score of -1.0. Note that by assigning a -1.0 to unobserved entities effectively "punishes" the scout team.

The "Tracker" in Figure 7-17 illustrates the amount of information a scout team would provide if each scout observed one enemy vehicle each at the beginning of the simulation and then "tracked" that entity to the conclusion of the simulation. Assuming no terrain occlusions, instantaneous message passing, and the third enemy not in vicinity of the tracked entities, the "Tracker" would receive an information score of (1.0 + 1.0 - 1.0) / 3 = 0.33 for each time unit.

The results show that the Soar+SVI agent provides more information upon initial visual contact (the slope in Figure 7-17 is steeper) and over a sustained period. Furthermore, on average, it sends more observation reports to the commander, indicating that the team has detected the enemy more frequently and that the overall architecture continues to be responsive as the agent is able to perform other functions (observe, report) in addition to imagery (Figure 7-18). The reason the Soar+SVI agent is able to reposition its team more effectively is that its analysis is more accurate. The Soar-SVI agent often under or overestimates the required adjustments resulting in the scout team missing critical observations.

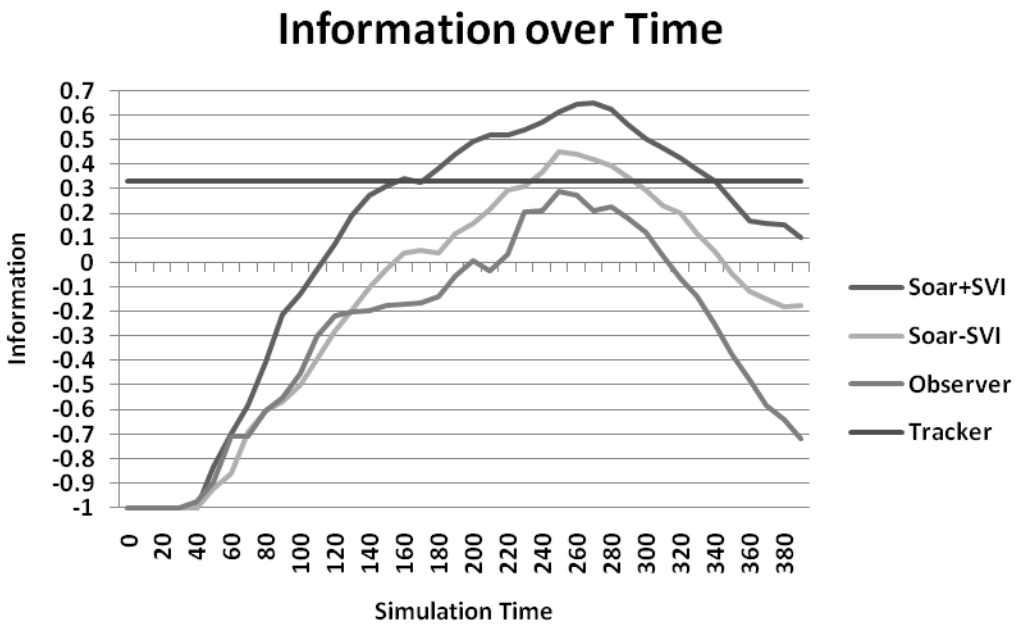



**Figure 7-17: Measure of Information over Time (Scenario-1)**

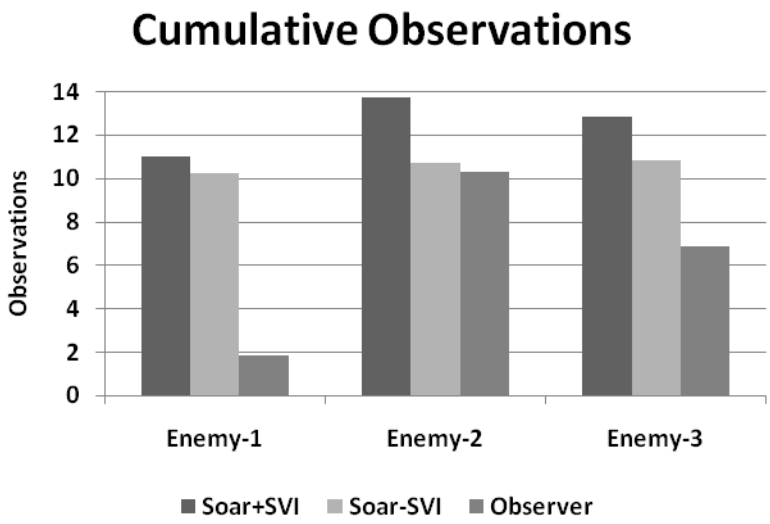
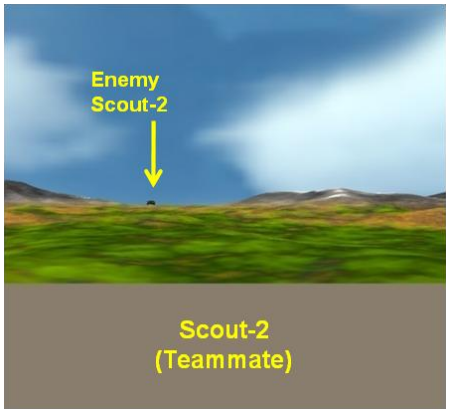



**Figure 7-18: Number of Cumulative Observations (Scenario-1)**

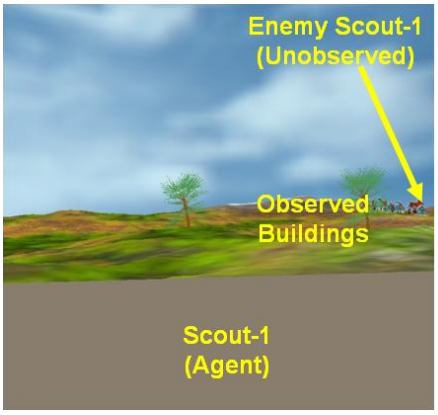
The purpose of the second scenario is to demonstrate the generality of the architecture and evaluate our hypothesis that the Soar+SVI agent analyzes the situation more accurately and thus, makes better decisions on where to orient its team. The second scenario uses the same terrain as in the first scenario, but the enemy's direction of attack is in the opposite direction and there are several buildings in the western sector that provide excellent coverage for the enemy scouts' movement (Figure 7-19a). The purpose of these reinforcing obstacles is to force the enemy to turn west before it can continue its movement north and determine if this sharp turn influences the agent's analysis.

In this scenario, Enemy Scout-2 initially maneuvers north/northwest and establishes visual contact with Scout-2 (Figure 7-19a&b). Once it makes contact, Enemy Scout-2 signals to Enemy Scout-1 to begin movement in a northwest direction. Enemy Scout-1 takes advantage of the buildings and terrain to remain concealed as much as possible from Scout-1's observation during a majority of the simulation (Figure 7-19c). Enemy Scout-3 randomly decides to follow Enemy Scout-1 or Enemy Scout-2. Note that in this maneuver the Soar+SVI and Soar-SVI agents begin with the same information as both establish visual contact with Enemy Scout-2 at approximately the same time (again, with some simulation variability). However, our hypothesis is that the Soar+SVI agent will perform a more accurate analysis taking into account the terrain and the building occlusions and adjust accordingly.

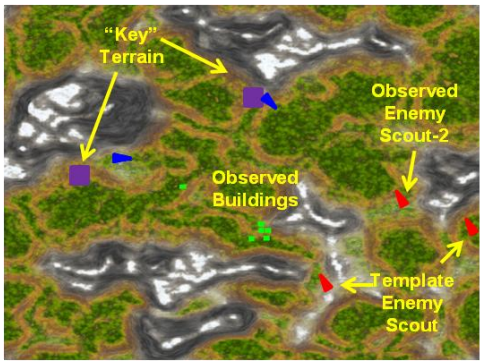

**(a) Actual Situation**




**(b) Teammate's View**




**(c) Agent's View**




**(d) Agent's Perceived Map / Imagined Situation**

**Figure 7-19: Scout Domain (Scenario-2)**

Figure 7-20 through Figure 7-23 illustrate the outcome. Figure 7-20 shows the cumulative information the agent provides to its commander over time. Initially, Soar+SVI and Soar-SVI provide similar amounts of information to the commander but at approximately simulation time 280, the Soar+SVI agent begins observing enemy movement again while the Soar-SVI agent, on average, does not. As Figure 7-20 shows cumulative information, to include Enemy Scout-2 which tends to cancel out the effects of observing the other two enemy scouts, Figure 7-21 and Figure 7-22 show the information that Soar+SVI and Soar-SVI agents provide on Enemy Scout-1 and Enemy Scout-3 respectively from simulation time 280 to 350. The results are more obvious for Enemy Scout-1 as Enemy Scout-3 sometimes follows Enemy Scout-2.

Why is there such a difference in the amount of information and number of observations? Figure 7-24 shows that visual imagery, and specifically visual depictive

manipulations, provides a more accurate representation of the enemy's hypothesized path. Using this information together with the generated view frustum that takes into account some terrain occlusions, the resulting analysis provides the Soar+SVI agent more accurate information to base its assessment.

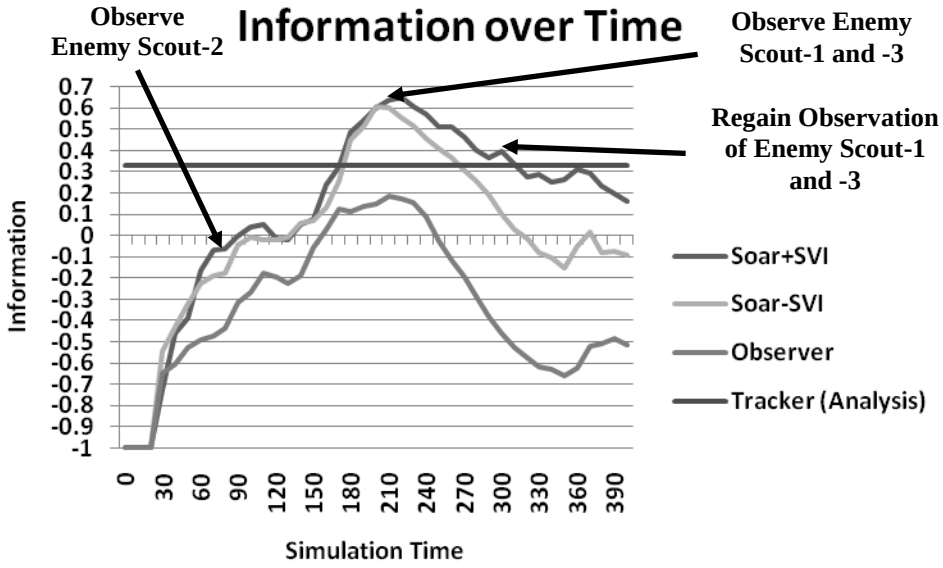



**Figure 7-20: Measure of Information over Time (Scenario-2)**

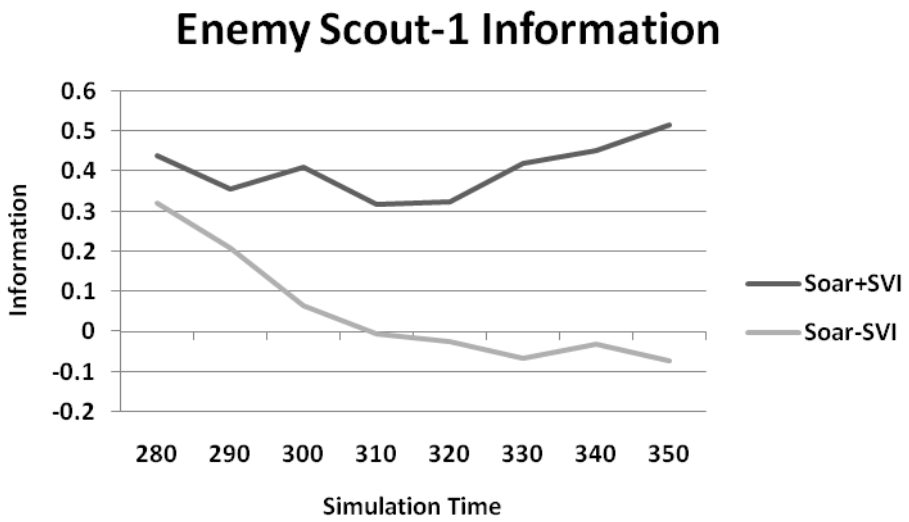



**Figure 7-21: Measure of Information on Enemy Scout-1 over Time (Scenario-2)**

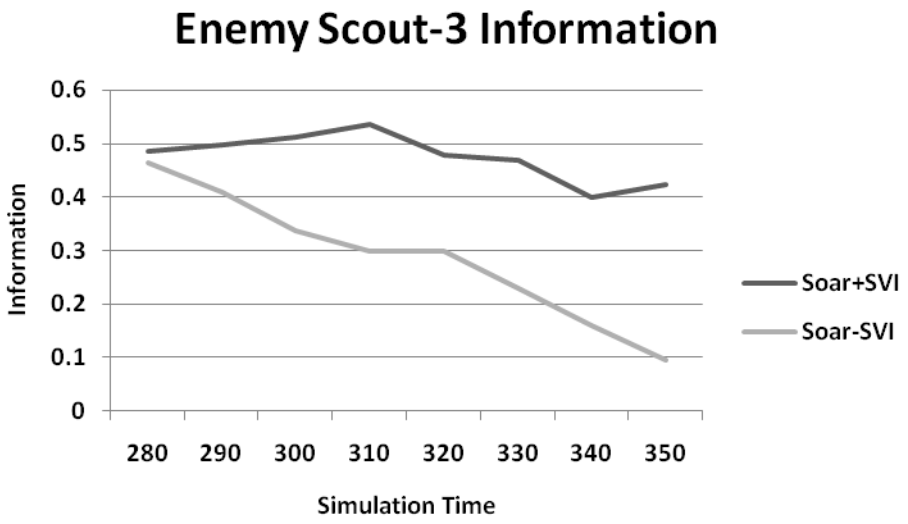



**Figure 7-22: Measure of Information on Enemy Scout-3 over Time (Scenario-2)**

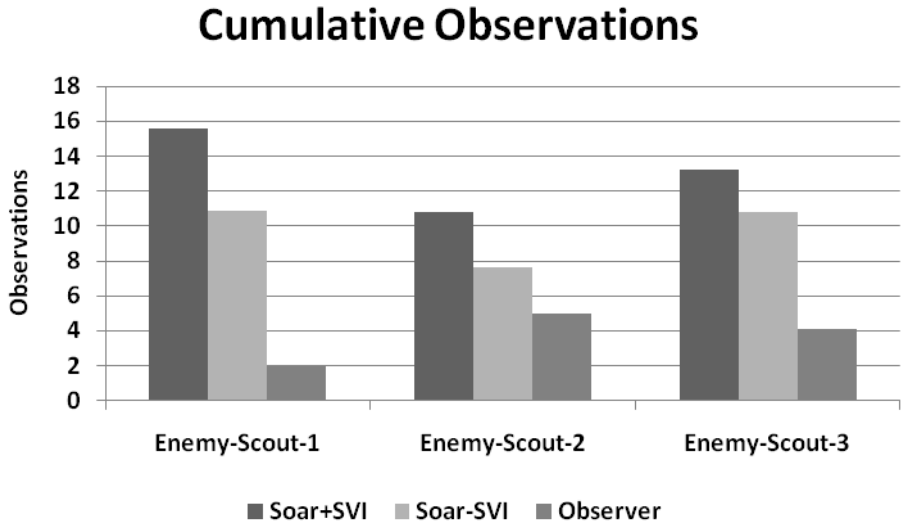
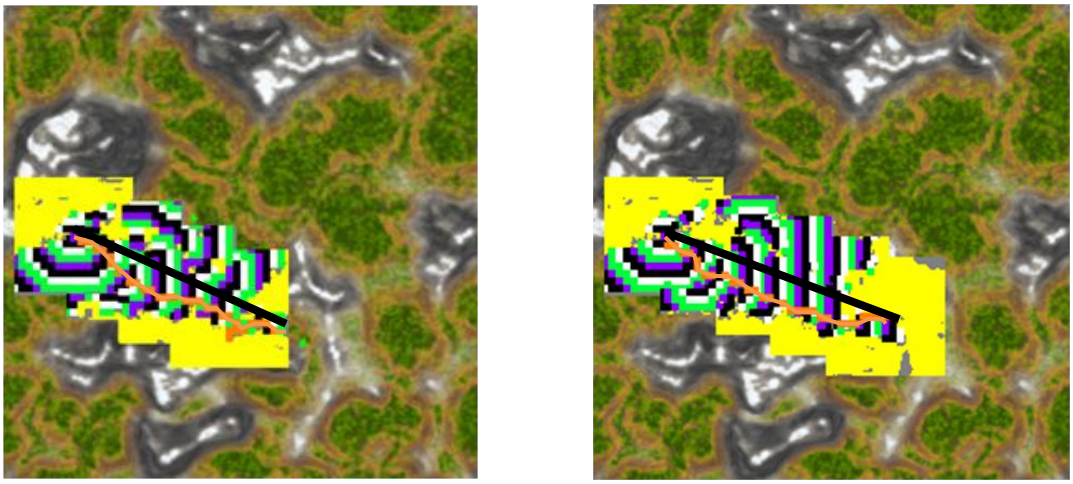



**Figure 7-23: Number of Cumulative Observations (Scenario-2)**



**(a)** **(b)**

**Figure 7-24: Two Example Paths Imagined During Scenario-2**
**The straight line reflects a quantitative spatial representation of the same path**

The gain in computational efficiency in processing the perceptual representations, functional capability, and problem-solving quality does not come without a cost. The resulting architecture is more complex and requires truth maintenance mechanisms to maintain consistency between the components when the agent stops imagining, either deliberately or because a more important perception interrupts it. Unlike the initial task knowledge of the geometry problem where the agent uses imagery for internal problem solving, Figure 7-25 suggests that because of the interaction with perceptual processing, additional task knowledge is required. Figure 7-25 shows the number of productions the Soar+SVI and Soar-SVI agents have in the Scout domain to include the "architectural" visual processing and imagery productions. Although the basic task knowledge (i.e. Observe, Decide, Act) is the same between the agents, the Soar+SVI agent requires more task productions because of its ability to perform analysis using imagery. This analysis includes adding and moving imagined objects, encoding depictive manipulations, and writing objects on the map—tasks that the Soar-SVI agent does not perform. The tradeoff then is that there is less task knowledge to perform "internal" cognitive tasks but more task knowledge to perform the "external" imagery operations and interact with perception.

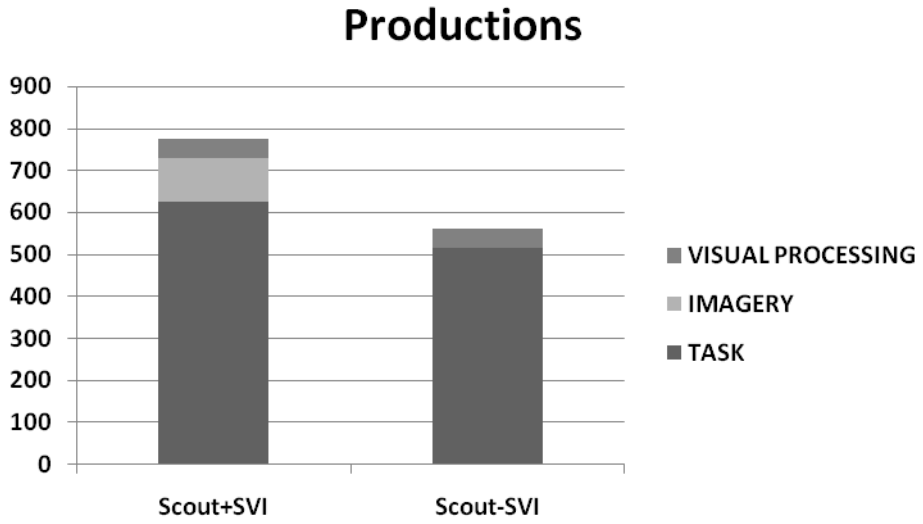



**Figure 7-25: Number of Productions in the Scout Domain**

### 7.4.5 Scout Domain Assessment

The Soar+SVI agent, by combining the symbolic, quantitative spatial, and visual depictive representations and then reasoning with them, can paint a relatively accurate picture of the situation for its commander sooner and for a longer sustained period of time than its counterpart Soar-SVI agent. Visual imagery enables it to provide a better, more fine-grained assessment than if it relied solely on spatial imagery. Furthermore, due to the number of observation reports, the results show that it is still able to perform other cognitive functions (receive and send reports, perceive the environment), demonstrating that the imagery system is working in conjunction with the complete cognitive system. Although SVI's depictive manipulations are not as responsive as we would like, they provide a significant computational advantage when compared to processing the representation with sentential algorithms and much more generality than a task specific implementation.

## 7.5 Lessons Learned

The evaluation demonstrates that the power of incorporating spatial and visual imagery mechanisms in a cognitive architecture emerges from the ability to combine their representations and reason with them. The experiments highlight tasks where spatial and

visual imagery fills in a missing details of a situation, recognizes novel shapes if not present visually, and assists an agent analyzing its actions before deciding and acting on them. As the results show, imagery is useful for inferring spatial relationships where metric information facilitates the reasoning between two or more objects (e.g. "sense of direction," orientation between the enemy and relevant key terrain) or detecting an object's spatial or visual properties (e.g. curves, enclosed spaces, path size, view coverage). In tasks where there are more than a few types of spatial properties or where visual features are not explicitly encoded, imagery provides an advantage from a computational, functional, and problem-solving standpoint. However, the advantages come at a cost to include architectural complexity and challenges in incorporating low-level components where the separation between knowledge and architecture is not always clear.

As we discussed in Chapter 2 (see section 2.2), each representation offers a functional and computational tradeoff. The Alphabet experiment and Scout domain highlight both these tradeoffs as visual imagery processing offers a functional advantage for reasoning about specific shapes but is less responsive than spatial imagery processing, suggesting that it requires greater capacity. Perhaps humans, when given a choice, tend to use spatial imagery in situations where speed is important. Maybe this observation is another reason why the mental imagery debate continues.

# Chapter 8

# Summary and Conclusion

Our research presents a synthesis of spatial and visual imagery, visual perception, and cognition. We demonstrate that it is computationally feasible to extend a general cognitive architecture with comprehensive mechanisms to support spatial and visual imagery processing to include quantitative spatial and visual depictive representations; shared mechanisms with vision; and incorporation of imagery's primary functions. Our empirical results and subjective assessment assert that for demanding spatial and visual tasks, the resulting architecture provides a computational gain and additional capability without trading off generality. As a summary of our work, this chapter reviews our major contributions, presents possible directions for future work, and concludes.

## 8.1 Research Contributions

The following list summarizes our major research contributions:

- **Integration of spatial and visual imagery's functional constraints (construction, transformation, generation, and inspection) in an implemented cognitive architecture (Chapters 3, 6, 7)**. We seriously consider the underlying behavioral and biological constraints in the design of the system. However, the integration focuses on functionality. We describe the representations and processes that are architectural and define the knowledge that is necessary to

create these representations and control the processing (e.g. visual object representations in VLTM, VS-STM, local spatial relationships in symbolic STM, a set of spatial and visual primitives, operators to control the high-level processing). As evidenced by the experimental results, we demonstrate these functions and their advantages in a working system.

- **Inclusion of a visual depictive representation and associated processing (Chapter 6, 7)**. An explicit mechanism for encoding and using the visual depictive representation is a major distinction between this work and other proposals. We demonstrate how the visual depictive representation provides the architecture with additional capability for recognizing visual features and spatial properties involving specific shapes. Furthermore, we show how it improves problem-solving quality when there is sufficient time for reasoning.

- **General mechanisms to support efficient processing of the quantitative spatial and visual depictive representation (Chapters 6, 7)**. Specialized processing is associated with each representation to gain efficiency. The inclusion of specific spatial and visual primitives facilitates communication between the processes in a general manner. We describe the challenges associated with achieving this generality and articulate the difficulties of determining how an agent (or human) decides when to use the appropriate representation. We suggest that functional capability, time, and desired accuracy are factors in this decision.

- **Description of the types of tasks where imagery provides a computational gain, additional functionality, or improved problem-solving quality (Chapters 2, 5, and 7)**. We describe and demonstrate several tasks where imagery processing is useful for reasoning because the task has many types or numbers of spatial or visual properties. These properties include direction, distance, orientation, size, topology, geometry, shape, and color. We suggest there are four general tasks where using imagery is useful to infer spatial and visual properties and demonstrate the first three tasks in our evaluation.

- Filling in missing details of a situation
- Recognizing novel shapes and spatial properties if not present visually
- Analyzing or rehearsing the outcome of an action before executing the action
- Replay of a previous event to inform a future decision

- **An initial computational theory describing the functional integration of visual perception with spatial and visual imagery (Chapter 3, 5, 6)**. We provide a computational theory of how imagery leverages the mechanisms provided by higher-level vision to facilitate its processing. Our theory includes the shared memories and processes between the two systems along with how bottom-up visual processing and top-down imagery processes coexist and mutually support one another.

- **Software engineering tools to support the evaluation and debugging of imagery components (Chapter 7, Appendix C)**. We built two tools to support the design, testing, and evaluation of the architecture. First, we integrate an SVI module with the SoarJavaDebugger (Figure C-6) to support viewing the contents in Visual LTM, the Object Map, and the Visual Buffer. Second, the simulation provides an initial attempt at defining an interface between an agent using Soar+SVI and an external environment.

## 8.2 Future Work

Based on our lessons learned from our evaluation (see section 7.5), we propose four possible directions for future work. There is some overlap between the research directions. We will explain each in more detail below.

1) *Push down*. Push the architecture closer to sensory input and low-level perceptual processing.

2) *Push up*. Explore the integration of spatial and visual imagery with Soar's declarative long-term memories (episodic and semantic).
3) *Push out*. Expand the capability of the current system by improving current functionality and expanding to other types of imagery processing such as motor imagery.
4) *Push in*. Build detailed cognitive models and compare to human data.

The first research direction is to expand our current perceptual theory by pushing the architecture closer to sensory input. One domain we would like to explore in more depth is robotics and specifically how cognitive processing, to include spatial and visual imagery, can provide a robot with higher-level reasoning abilities. Paramount in this exploration is an understanding of how our theory of perceptual processing would incorporate typical robotic sensors (e.g. light detection and ranging, stereoscopic video images, global position system, etc.) and how imagery may prime robotic effectors (motor imagery). Kuipers (2000) spatial semantic hierarchy, Yeap's and Jefferies' (1999) absolute space representation, and Ullman's (1996) object recognition schemes provide some insights here.

This research direction could also consider the role of imagery in top-down visual perception. For example, imagery may support the recognition of an object when bottom-up visual perception initially fails to recognize the object but suggests possible candidate objects (i.e. partial matches). Top-down visual processing using imagery may generate a visual depiction of each candidate object transformed from their stored, canonical orientation to an orientation congruent with the unrecognized object. The architecture attempts to recognize the perceived object by comparing its extracted visual features with the features extracted from the visual depiction of the imagined, transformed object. Relevant research questions include how many candidate objects to consider and how to determine when the transformed imagined object is "congruent" with the orientation of the unrecognized object.

Imagery processing may also support object permanence. Object permanence is the memory of a visible object that disappears (through motion) behind another object. The architecture may realize such behavior by maintaining the object's quantitative

spatial representation for some time after it visually disappears from the perceived scene. If the object was in motion the architecture combined with knowledge about the object's motion model (discussed below), may simulate object movement until it reappears from behind the occluding object or decays from memory. An important research question to address is how long the object should persist after it disappears from the scene.

The second approach is to explore how imagery integrates and interacts with episodic and semantic memories. As an agent has experiences, episodic learning may store certain aspects of the experience, including perceptions, internal state, and resulting action, as structures in memory. These structures may include symbolic, quantitative spatial and, possibly, a few specific visual depictive representations. At some time in the future when the agent recalls the experience to inform a decision, it may use imagery processing to replay the experience and infer spatial and visual properties that perhaps it did not explicitly encode or use as part of its original decision-making. This imagery replay capability also presents an opportunity for "offline" learning as the agent can reason about new spatial and visual properties that it did not attend to during the actual situation (e.g. Alice was seated to the left of Bob at the party last night).

Semantic memory stores knowledge that is more general rather than specific instances. In the context of imagery, this memory is useful for encoding the local spatial and visual properties that imagery uses to construct the scene. For example, the fact that a place setting has a fork, plate, knife, etc., and that the knife is right of the plate or that the enemy typically configures itself in a particular formation (i.e. doctrine) are examples of semantic memories. Note that imagery can help keep the encoding of semantic or episodic memories compact, as only the local spatial relationships between the objects and their explicit visual properties need to be stored. Reconstructing the situation with imagery enables the inference of global spatial relationships and visual properties not explicitly encoded.

Together imagery and these long-term declarative memory mechanisms have potential to lead to more informed reasoning. For example, in the Scout domain, an agent may initially construct the imagined parts of its scene from its semantic knowledge of the enemy and the terrain. As the agent acquires more experiences, it may adjust its templates. The enemy may change the spatial characteristics (distances, directions, and

orientations) of their tactics or maneuver through an area the agent previously thought was impassable. With these experiences, the agent may then construct and analyze its imagined scene using the adjusted templates from its episodic memory.

The third research direction involves extending our current work by improving the communication primitives and algorithms for spatial and visual properties, refining our notion of the Visual Buffer's attention window, and using motion models in addition to one-step transformations to simulate movement. We have designed the system to incorporate the basic spatial and visual properties listed in Figure 6-9 thru Figure 6-11. Although we offer a small contribution in this work with this list of properties, there does not appear to be a cohesive theory detailing the spatial and visual primitives that humans use in reasoning. There are researchers (Biederman, 1987; Cohn et al., 1997) who offer theories for a specific spatial or visual property type. However, without a coherent theory there remain challenges in designing a general-purpose architecture because, as we have discussed, the interpretation of these properties tend to constrain each other (e.g., constructing a topological relationship between two objects requires knowledge about their specific shape and orientation, symmetry may or may not be a primitive feature, etc.).

Related to this issue are the factors an agent uses to determine the representation to use for reasoning. The factors we suggest are the following:

1) Functional capability. I am detecting curves so I have to use visual imagery.
2) Speed/Accuracy tradeoff. If both representations provide a result but one is more accurate and I have time, then use the visual representation.
3) Number and types of spatial and visual properties. The greater the number and types, the more likely reasoning requires a visual depictive representation.

A future research effort then is to continue to review the literature in an effort to refine and improve these low-level primitives and investigate how learning mechanisms may use the factors, such as what we suggest above, to assist in choosing the appropriate representation.

The attention window proves useful in improving the efficiency of the depictive manipulations. However, its design is brittle as it has a fixed size and shifts only in a linear direction based on procedural knowledge. A more flexible approach is to allow the

attention window to “grow”/“shrink” and also shift based on a visual cue (i.e. shift to the red object). For example, in the Scout domain if the terrain is very restrictive, it will impede the distance field flood. An architectural mechanism could possibly detect this “impasse” and either inform cognition or automatically expand or shift the attention window so that processing can continue. The major research questions for this direction is how does the architecture detect and signal this type of “impasse” and how does the resize and/or shift occur-- automatically or deliberately through procedural knowledge?

One interesting phenomenon in humans is their use of motor imagery to rehearse potential actions. Such a priming of the motor system, rather than executing a one-step transformation, appear to take into account factors such as force and torque using motion models of the particular subsystem (Grush, 2004). The use of motion models for simulating the motion of other objects, such as the trajectory of a thrown baseball or the movement of a vehicle is also applicable here. Note that these motion models may be in the form of a quantitative, dynamical system or a set of depictive manipulations. In these types of transformations, time is a key parameter, as the architectural processing must know how long to run the simulation. A few of the major issues for the incorporation of motor imagery and, specifically motion models, is to determine what memory structure(s) store these models, how they originate and dynamically adjust, the granularity of time in simulating the model, and insuring that the simulation of the model is not an unconstrained computation. Wintermute and Laird have begun to investigate these issues (2008).

The final research direction is to explore detailed cognitive models and attempt to match human data. As we previously alluded to in the Alphabet Experiment, modeling the details of low-level perceptual processing dominates this goal and thus relates back to our first research direction of attempting to push the architecture closer to sensory input. However, it also relates to our other research directions, as there are some higher-level issues to address such as continuing to flush out the spatial and visual primitives to refine what knowledge should be “hardwired” into the architecture rather than being encoded in a declarative or procedural memory.

One of our specific ideas for this research path is to run an experiment with human subjects performing the same task as the agent in the Scout domain. During the

experiment, we would capture fMRI and eye-tracking data, measure response times for major decisions (i.e. when the subject decides on a course of action), and conduct post-experiment interviews in an effort to understand how the individual solved the problem. Assuming this dataset provides us with enough evidence that some form of imagery is being used (e.g. parietal or visual cortex activation, subject says they "imagined" the situation), we could then start building models and try to match the human data in this domain.

## 8.3 Conclusion

Past research in cognitive architectures has primarily taken the stance that amodal, symbolic representations are sufficient for thought. This research expands this notion by beginning to link perceptual-based representations with cognition. This union provides functional and computational advantages for reasoning about spatial and visual properties. The new capabilities of the resulting architecture that includes both Soar and its Spatial-Visual Imagery (SVI) component emerges from its ability to combine multiple representations and reason with them. Soar's symbolic memories and processes provide the building blocks necessary for high-level control in the pursuit of goals, learning, and the encoding of amodal, symbolic knowledge sufficient for general, abstract reasoning. SVI encompasses the quantitative spatial and visual depictive representations and processing specialized for efficient construction and extraction of spatial properties and visual features not encoded as symbols. Together these mechanisms are necessary if we hope to achieve general intelligence.

# Appendices

# Appendix A

# Supporting Behavioral and Neuroimaging Experiments

While modeling human performance on these tasks is not a goal of this research, the following experiments[29] motivated our theory, design space constraints, resulting architecture, and evaluation domains. Each experimental description includes the reference, the described imagery functionality (*construction, transformation, generation, inspection),* its task type (spatial or visual imagery), a summary of the experiment, the relevant results, and a short discussion.

## A.1 Image Units and Relations

a. Reference. (Kosslyn et al., 1983).
b. Functionality. Construction, Generation, Inspection.
c. Task Type. Visual.
d. Summary. The experimenters gave the subjects specific instructions on how to encode geometric objects (Figure A-1) using either coarse or fine object parts (see figure for definition of object parts) and their spatial relationships. After the subjects indicated that they had visualized the image (by pushing a button), they were probed for a specific feature. For example, in the first figure they might be asked if they see a "bow," a "cross," or whether the object is symmetrical about

[29]The "island scan" experiment discussed in Chapter 2 is another experiment that influenced our theory.

the vertical axis. The experimenters recorded the time to visualize the image and response to the probes.

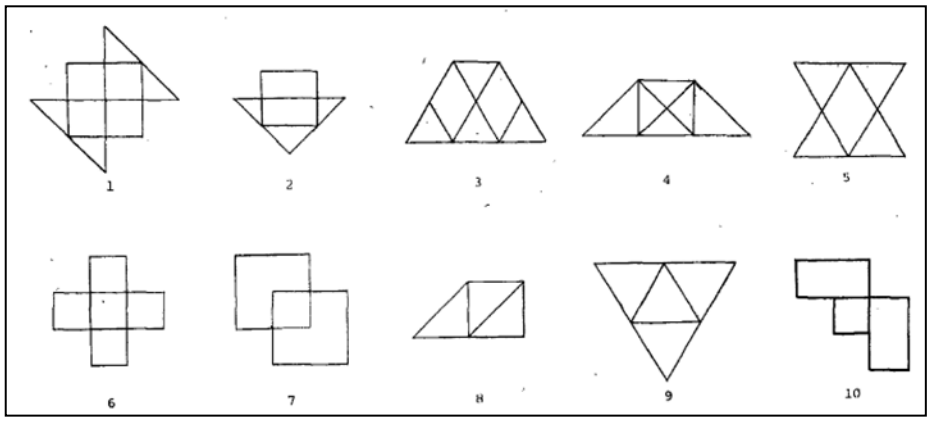



**Figure A-1: Stimulus Patterns**

**(1) 2 triangles, 1 square vs. 4 triangles, 4 squares; (2) 1 triangle, 1 square vs. 2 rectangles, 3 triangles; (3) 1 hexagon, 2 triangles vs. 2 diamonds, 4 triangles; (4) 2 triangles, 1 square vs. 6 triangles; (5) 2 triangles vs. 4 triangles, 1 diamond; (6) 2 rectangles vs. 5 squares; (7) 2 squares vs. 2 L's, 1 square; (8) 1 square, 1 parallelogram vs. 3 triangles; (9) *2* triangles vs. 4 triangles; (10) 2 L's vs. 2 rectangles, 1 square.**

e. Results. Subjects who encoded the figures with more parts took more time to generate their visual images of the shape. Subjects required less time to see a pattern in the image when the feature was congruent with the original description. The authors concluded that (1) people construct visual images by amalgamating an object's parts. The addition of each part to the image requires time. (2) In addition to metric shape information, people use descriptive (symbolic) information in constructing images; and (3) the ease of visualizing and inspecting an object depends on how many parts composes it and its symbolic description.

f. Discussion. This experiment corroborates the theory that images are constructed incrementally by adding parts. It also highlights that the descriptive or symbolic representation is dependent on how one originally encodes the object. The symbolic description of the object, or its super ordinate category, will not include all of the object's spatial and visual properties such as whether it has an enclosed space or contains four squares. Thus, there is the necessity to visualize and inspect the object when attempting to recall these properties. The second capability that

this experiment demonstrates is extracting emergent objects (bow, cross) or features (symmetry) by composing known objects in novel ways.

## A.2 Detecting Implicit Object Features

a. Reference. (Thompson et al., in press).

b. Functionality. Generation, Inspection.

c. Task Type. Visual.

d. Summary. The researchers' purpose was to gather evidence for the depictive nature of representations during visual imagery and compare the underlying mechanisms used to those used in visual perception. For both the visual perceptual and visual imagery trials, they compared the ease of judging shape properties for uppercase letters of the English alphabet. Some of the shape features were propositionally explicit (and thus, immediately accessible to the verbal system) while others were properties that the subject had to infer because they were not explicitly encoded (i.e. in a symbolic representation). Their hypothesis was that that the mechanisms enabling visual imagery are similar to visual perception.

The researchers first determined what features of an uppercase letter are explicit by having a group of participants classify each visual appearance of each letter and describe its shape properties as though they were talking to a blind person who someday may be able to see the letter. For example, the letter 'A' may have been described as one diagonal line slanted to the left, connected at the top to one diagonal line slanted to the right, with one horizontal line centered between the two diagonal lines. The researchers considered the feature explicit if the subjects mentioned it more than 50 percent of the time and not explicit if it was mentioned less than 5 percent of the time. They discarded features mentioned between 5-50 percent of the time. The most represented explicit features mentioned were "line," "diagonal line," "curve," and "semi-circle." The features deemed not explicit (mentioned less than 5 percent) were "enclosed space" and "symmetrical."

Next, the researchers split a different set of subjects into two groups: a perception group and a visual imagery group. The task for each group was similar (Figure A-2). Each group first heard a letter. Then the participants would either see

the letter on a display (visual perception) or visualize (imagery) the letter based on the previous instruction. Next, the participants heard the feature to evaluate (e.g. "enclosed space," "curve," "diagonal line," "vertical symmetry," "horizontal symmetry") and responded by pressing a "yes" or "no" key as quickly as possible. The experimenters measured the subjects' response times (RT) and error rates.

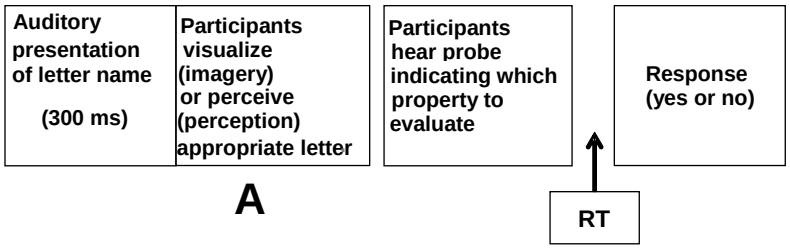



**Figure A-2: Trial Format for Detecting Alphabet Letter Features**

e. Results. There was no interaction between the visual mode (perception, imagery) and method of encoding indicating that visual perception and visual imagery rely on the same mechanisms. Additionally, explicitly encoded features (i.e. curves and diagonal lines) required less response time than implicitly encoded features (symmetry and enclosed spaces) for both visual perception and visual imagery. The "symmetry" features also produced the largest error rate.

f. Discussion. This experiment was the basis for one of our evaluations. What it demonstrates is that visual perception and visual imagery share similar mechanisms and that there are some object features humans do not explicitly encode as a descriptive, symbolic representation. Therefore, it highlights the visual imagery capability of being able to reacquire these features.

## A.3 Imagery Transformations

a. Reference. (Shepard & Metzler, 1971).

b. Functionality. Transformation, Inspection.

c. Task Type. On a spatial and visual spectrum, this task falls somewhere in between. Our hypothesis is that the spatial representation is used to perform the transformation, and a visual depictive representation is required to recognize if the two objects are the same.

d. Summary. Shepard and Metzler showed subjects pairs of three-dimensional, non-standard objects and asked them to determine if the objects were the same shape (Figure A-3). Some pairs were identical but with one of the objects rotated at a different angle than another. Other pairs were mirrored reflections of one another so could not be brought into correspondence by a rotation. After shown a pair of objects, subjects pulled a right-hand lever if they thought the objects were congruent and a left-hand lever if they did not think they were congruent. Response times were measured.

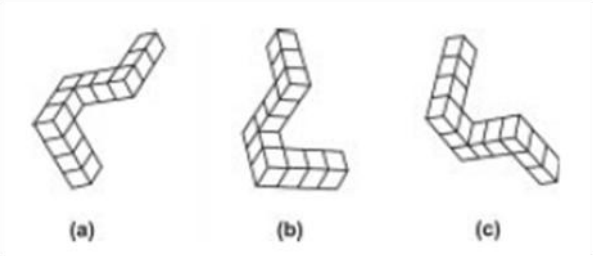



**Figure A-3: Mental Rotation Shapes**

e. Results. Shepard and Metzler measured response times for each subject and found that the response times were linear with the rotation angle. The subjects' introspective reports claimed that in order to make the comparison they had to "mentally rotate" one of the objects. These two pieces of evidence led them to hypothesize that there is some sort of an imagined transformation process in three-dimensional space.

f. Discussion. We are using this experiment to highlight not only the phenomenon of being able to rotate three-dimensional objects but also as motivation for transformations of quantitative spatial and visual depictive representations in general.

## A.4 Combining Perception and Imagery

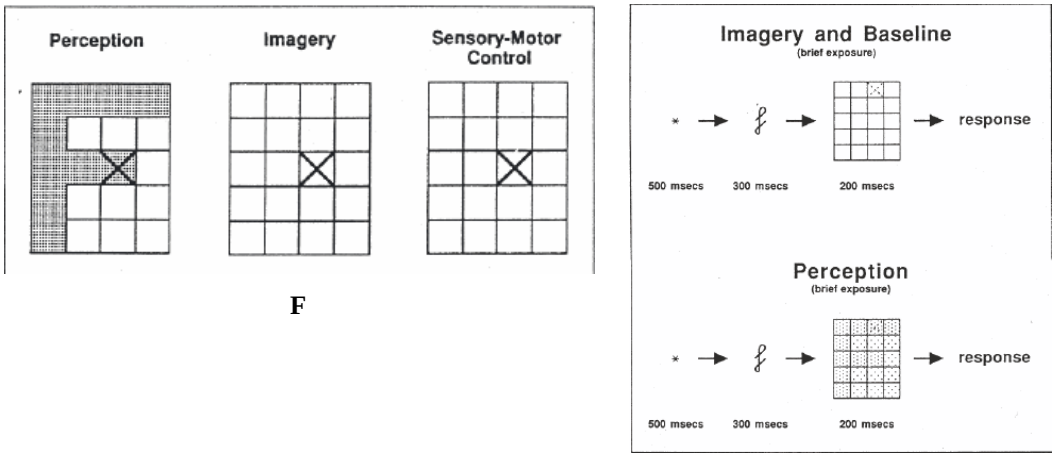
a. References. (Kosslyn et al., 1993; Podgorny & Shepard, 1978)

b. Functionality. Construction, Generation, Inspection.

c. Task Type. Visual.

d. Summary. Experiment originally devised by Podgorny and Shepard to measure the functional correspondence between visual perception and visual imagery. There were two experimental groups. Each experiment started by displaying a two-dimensional 5x5 grid to the subjects (Figure A-4a). The perception subjects viewed one or two English letter(s) in the grid. A visual probe in the form of one or more dots would appear in the grid and the subjects responded as quickly as possible as to whether a dot fell on the letter(s). Podgorny and Shepard measured response times and recorded other factors such as number of dots and distance (number of grids) of the dot(s) from the letter.

The second imagery group, rather than viewing the letter(s), were instructed to visualize it in the grid and press a pedal when they had the letter imagined. At that time, the dot(s) probe appeared, and as in the first case, the subjects indicated their response as quickly as possible as to whether a dot fell on their imagined letter.




**(a) Subjects saw a letter in grid (perception task), visualized letter based on a "script cue" (imagery task), or waited for 'X' mark to be removed (sensory-motor task)**

**(b) Subjects saw script cue, then a perceptually degraded upper case version of cue (perceptual task), and a degraded 'X' (both tasks) for the amount of indicated time**

**Figure A-4: Identify the 'X' On / Off the Letter**

Kosslyn et al. (1993) extended the experiment in several ways, a few of which are described here. First, they used PET (Positron Emission Tomography) to measure the emissions from a radioactively labeled chemical injected into the subject's bloodstream. The PET data produces two- or three-dimensional images

of the distribution of the chemicals throughout the brain and provides an indication where brain activity is occurring. Second, rather than seeing one or more dots, subjects would see a single 'X' in one of the grids (Figure A-4a). Half of the time the 'X' fell on the letter; half of the time if fell off. Additionally, half of each type (on/off) was drawn near the segment of a letter that was thought to be imagined early in the visualization sequence. The other half was drawn closer to those segments thought to be imagined later. The purpose of this experiment variability was to test Kosslyn's hypothesis that humans build the image of an object (in this case a letter) by composing the parts, one part at a time. Third, Kosslyn included a control group (the sensory-motor group) that simply saw an 'X' in the grid and responded when the 'X' disappeared. The purpose of this control group was to exclude the activated brain areas and response times that sense the 'X' and control the motor response.

Finally, Kosslyn et al. ran another experiment to induce the recall of visual memories (Figure A-4b). They hypothesized that the first task may not access visual long-term memory because the tasks were what they call "attention based imagery." In this second task, subjects in the perceptual group were shown a script letter, followed by the letter and the grid. Then the 'X' appeared for 200 ms in a degraded form. In the imagery group, only the grid and 'X" appeared. The idea was that the task would no longer be based solely on attention because the subjects could not just fix attention on that region. Rather, they would have to recall the letter and the grid from visual memory for both perception and imagery.

e. Results. In Podgorny and Shepard's experiment, they found that the response times varied with the number and locations of the dots (whether they were on or off the letter) and, as expected, the response times for the imagery group were longer than for the perception group. However, the factors influencing the variance in response times had a similar affect for both perception and imagery leading them to conclude that perception and imagery use similar mechanisms.

Kosslyn's group found subjects required less time to evaluate probes where the 'X" fell closer to the line segments of a letter believed to be added earlier in the image construction process. The neurological evidence indicated

greater activation of visual cortex during visual imagery than during perception for both tasks. In addition, for Task 2 they discovered activation in other areas, such as the dorsolateral prefrontal cortex (DLPFC), occipital-temporal pathway, and parietal regions.

f. Discussion. This experiment features the intersection between vision and imagery to included shared memories (visual cortex, temporal lobe, parietal lobe) and processing. It also serves as an example of the ability to "perceive/imagine/re-perceive" behavior. That is, imagined spatial and visual representations can augment visual perception to aid in the decision-making process. The grid and the dot/'X' arrive in the visual buffer from vision and are not generated from imagery. The subject adds an imagined letter to this perceived scene. This is similar to our Scout domain where the agent augments its map by imaging different objects and features on it and then re-perceives the image.

## A.5 Map and First-Person Perspective Recon

a. References. (Mellet et al., 2000)
b. Functionality. Construction, Inspection.
c. Task Type. Spatial.
d. Summary. The experiment tested the two major sources of information to build a topographic representation of an environment, actual navigation within the environment (route perspective) and map learning (survey perspective). The experimenters used positron emission tomography (PET) to compare the neural substrate of the topographic representation built from these two modes.

Mellet et al. broke subjects into two groups: a "route perspective" and a "survey perspective" group. The experiment had three phases: (1) learning, (2) training, and then (3) testing. During the learning phase, the "route perspective" group walked through a park they had never seen before (Figure A-5). An instructor led the walk by taking subjects to seven key landmarks (statue, tower, lake, etc) in the order they later would be expected to recall. After the instructor led iteration, the subjects repeated the walk two more times following the same path.

The “survey perspective” group’s learning phase consisted of studying a map of the park with the same landmarks annotated on the map as seven different colored dots. A path linked the dots. Experimenters then ran subjects through a series of seven slides showing each landmark in the same order that had been presented to the walking group. The experimenter told the subjects the color of the dot on the map that represented the landmark so the subjects could associate the dot to the landmark. To insure the subjects learned the map, they were required to pinpoint each dot location on a blank map at the end of the learning phase.

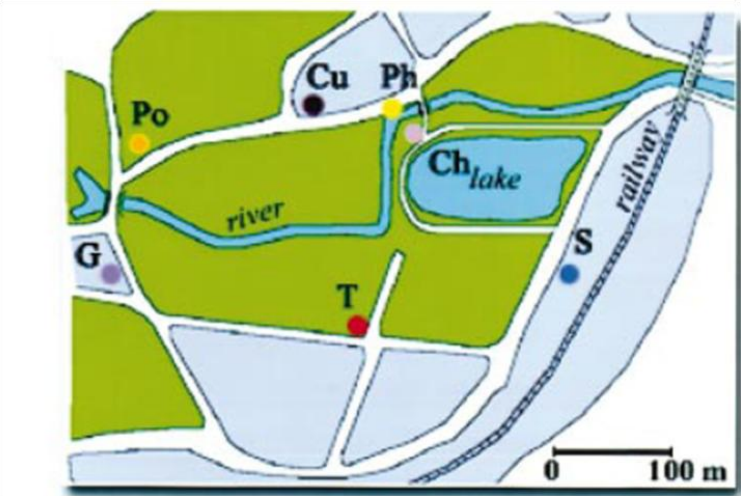



**Figure A-5: Park Map Used in Mellet et al. Experiment**

During the training phase (3-4 hours before the testing phase), the route perspective group trained on a mental navigation task by being presented with two landmark names ("gas station," "phone box") and then were instructed to visualize the walk between the two locations by mentally simulating it. When the subject imagined their “arrival” at the second location, they pressed a key. The map group similarly trained on the mental navigation task by visualizing the map as accurately as possible including the seven dots. They were then presented with two dot colors (“red,” “blue”), and had to imagine a laser dot following the path segment on the original map between the two dots. Once the second dot was reached, the subject pressed a button triggering the release of the next pair of dots. The training consisted of three sessions with each session including the mental

navigation between five pairs of landmarks. Path segment length varied between 48 and 172 meters. During the testing phase, the experimenters administered a PET scan while the subjects in both groups were either (1) resting with eyes closed or (2) mentally navigating as described above.

e. Results. The right hippocampal and intraparietal sulcus were active in both groups indicating the spatial imagery component of the task. There was no activation observed in the visual cortex.

f. Discussion. These two different tasks highlighted differences in encoded material (2D vs. 3D) and task demands (egocentric vs. allocentric view). However, both showed similar activation. Hippocampus is associated with what we call episodic memory indicating possible interaction between the imagery system and episodic memory. Task properties included (1) recall of spatial locations and landmarks, (2) maintenance of spatial relationships in the scene, and (3) mental simulation of displacement from one location to the next. The experiment has some similarities with the Scout domain as the agent has to imagine an enemy's current location by simulating its movement.

# Appendix B

# Algorithms

This appendix describes specific algorithms used for detecting curves and manipulating depictive representations. For the purposes of this appendix, an image, ***I,*** consists of a *spatial domain* **X** and an ***F****-value* set. ***X*** is a topological space consisting of points and the topology providing the notion of connectivity. For example, a two-dimensional point, **x**, is described as $(x_i,y_i)$ where $x_i$ and $y_i$ describe the location of the point in a two-dimensional space. The graphical representation of a point set, $\boldsymbol{X} = Z_m \times Z_n$, is shown in Figure B-1. An ***F****-value* set is a set of possible values together with a finite set of operations. In this discussion, we are concerned with integer and real (float) value sets. The image, ***I***, is then represented by a data structure ***I*** = s and an element of ***I***, (**x, *I*(x)**) is called a *picture element* or *pixel*. The first coordinate, **x**, is the pixel location and the second coordinate, ***I*(x)**, is the pixel value of ***I*** at location **x** (Ritter & Wilson, 1996).

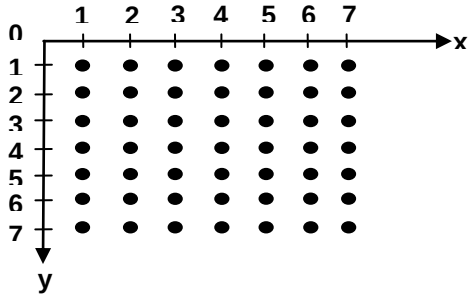



**Figure B-1: Point Set X =** $Z_m \times Z_n$

## B.1 Hough Transform

To detect lines and curves in an image, we use the Hough transform (Mat Jafri & Deravi 1994; Olson, 1999; Ritter & Wilson, 1996). The Hough transform is a "voting" algorithm that maps edge points in an edge-detected image to parameters in a parameter space. That is, given an edge point, **x** from the edge point-set, **E**; a set of parameters, Ω, that describes the curve; and either an analytical function, ***f***, or a lookup table that parameterizes the curve; the Hough transform, *h*, is described mathematically as

$$\mathbf{x} = (x_0, y_0)$$
$$\Omega = (\omega_0, \omega_1, \cdots \omega_N)$$
$$f(\mathbf{x}, \Omega) = 0$$
$$h : (\mathbf{x}, \Omega) = \begin{cases} 1 \; if \;\; f(\mathbf{x}, \Omega) = 0 \\ 0 \quad otherwise \end{cases}$$

Each edge point in the edge-detected image has an opportunity to "vote" on one or more sets of parameters. The algorithm collects votes in an array of counters, called the accumulator or vote array. The array is a discrete partition of a continuous multidimensional space spanning all feasible parameter values defined by either the analytical equation or a shape lookup table. Larger counts in the vote array indicate a higher probability that the parameter indices of the array are the parameters of the shape in the image. A threshold algorithm must determine what set of parameters, if any, are a representation of the sought-after shape. Computer vision researchers have used the Hough algorithm to detect lines, circles, ellipses, and other non-analytical shapes that have an associated lookup table describing the shape.

There has been little published on using the Hough Transform to detect parabolic curves, but it appears to be a good fit for finding general curves in any orientation. We use Mat Jafri and Deravi's (1994) algorithm and extend it to detect false positives. We can define a parabola as a locus of points equidistant to a fixed point called the focus, ***F***, and a fixed straight line called the directrix, ***d***. Figure B-2 shows a parabola in its "canonical" form with its vertex, $(x_0, y_0)$ at the origin. Its focus, ***F***, is on the x-axis a distance, ***a***, from the origin, and its directrix, ***d***, is parallel to the y-axis at a distance, ***a***, from the y-axis. This canonical form of a parabola has an equation of

$$4ax = y^2 or\ x = \frac{y^2}{4a} \qquad (1)$$

where ***a*** represents the length between the focus and the vertex. In the canonical form the focal point is located at (a,0), and the focal length, ***a***, defines the "curvature" or the width of the curve. The default value for ***a*** given the equation $x = y^2$ is 0.25. As ***a*** approaches 0 the parabola becomes "skinny" to where the two ends would eventually converge into a line. As the focal length approaches infinity, the parabola widens. At infinity the curve straightens into the line, x = 0 (the y-axis).

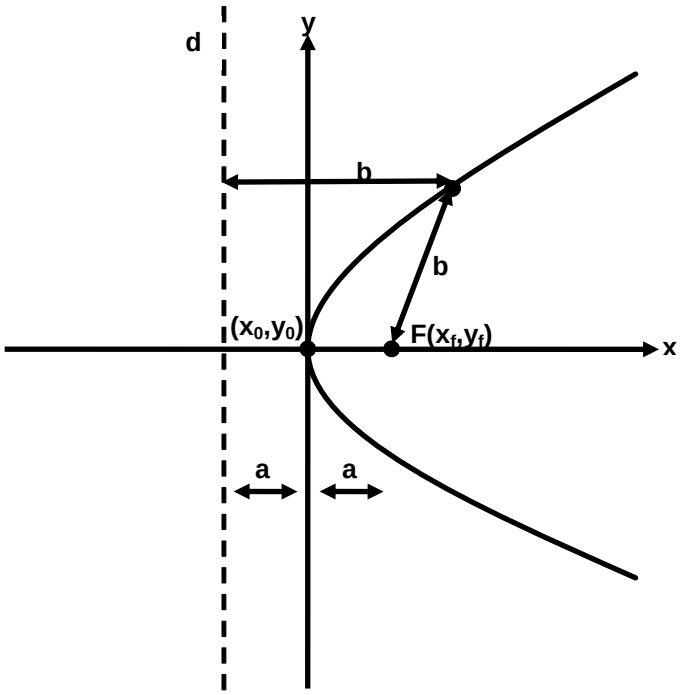



**Figure B-2: Parabola in Canonical Form**

Figure B-3 shows a parabola in its general form with the vertex translated ($x_0$ $y_0$) from the origin and a counterclockwise angle of rotation, Θ, from the x-axis. We can describe a translated parabolic curve without rotation as the following equation:

$$4a(x - x_o) = (y - y_0)^2 \qquad (2)$$

When we apply the angle of rotation equation (2) becomes the more generalized equation:

$$\begin{bmatrix} \cos\Theta & -\sin\Theta \\ \sin\Theta & \cos\Theta \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x\cos\Theta - y\sin\Theta \\ x\sin\Theta + y\cos\Theta \end{bmatrix} \text{ therefore,}$$

$$4a[(x\cos\Theta - y\sin\Theta) - (x_o\cos\Theta - y_0\sin\Theta)] = [(x\sin\Theta + y\cos\Theta) - (x_o\sin\Theta + y_0\cos\Theta)]^2 \quad (3)$$

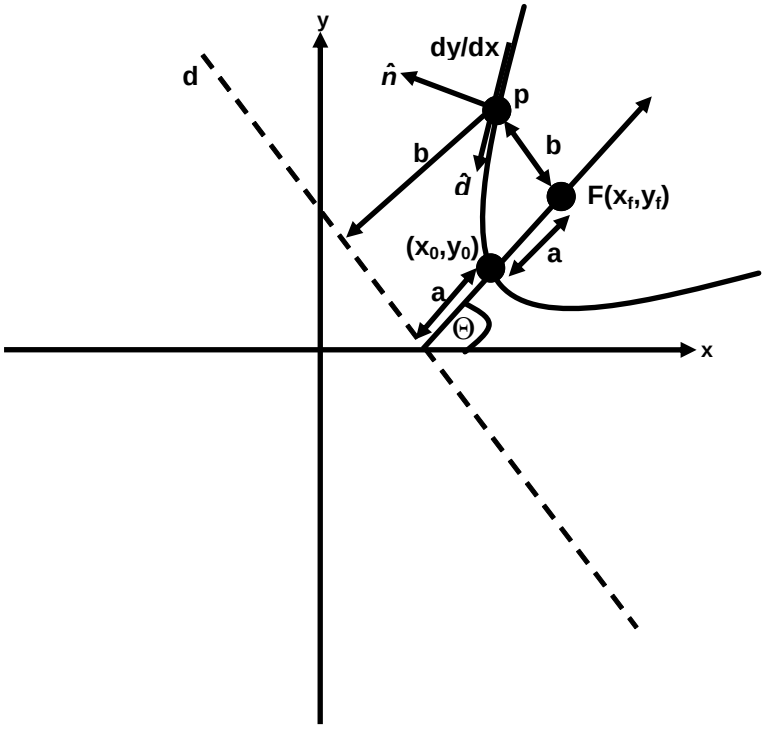



**Figure B-3: General Parabola**

If we then differentiate y with respect to x in equation (3), we obtain the following equation:

$$4a[\cos\Theta - {dy}/{dx}\sin\Theta] =$$

$$2*[(x\sin\Theta + y\cos\Theta) - (x_o\sin\Theta + y_0\cos\Theta)]*[\sin\Theta + {dy}/{dx}\cos\Theta]$$

$$a = \frac{[(x\sin\Theta + y\cos\Theta) - (x_o\sin\Theta + y_0\cos\Theta)]*[\sin\Theta + {dy}/{dx}\cos\Theta]}{2*[\cos\Theta - {dy}/{dx}\sin\Theta]} \quad (4)$$

Substituting (4) back into (3) results in the equation:

$$\frac{2*[\sin\Theta + {dy}/{dx}\cos\Theta]*[(x\cos\Theta - y\sin\Theta) - (x_o\cos\Theta - y_0\sin\Theta)]}{[\cos\Theta - {dy}/{dx}\sin\Theta]} =$$
$$[(x\sin\Theta + y\cos\Theta) - (x_o\sin\Theta + y_0\cos\Theta)]$$

Simplifying the equation:

$$k = \frac{2*[\sin\Theta + {dy}/{dx}\cos\Theta]}{[\cos\Theta - {dy}/{dx}\sin\Theta]}$$
$$k*[(x\cos\Theta - y\sin\Theta) - (x_o\cos\Theta - y_0\sin\Theta)] = [(x\sin\Theta + y\cos\Theta) - (x_o\sin\Theta + y_0\cos\Theta)]$$

Solving for $y_0$

$$y_0 = \frac{lx_o - lx + my}{m}$$

where

$$l = k*\cos\Theta - \sin\Theta$$
$$m = \cos\Theta + k*\sin\Theta$$

(5)

Equation (5) serves as the analytical function, **f**, which we use to determine the possible set of parabolas where a particular edge pixel may fall. The parabola is parameterized by $\Omega = (x_0, y_0, \Theta)$. To calculate the slope of the line tangent to the parabola at a pixel edge, p, (dy/dx in equation (4)) we use the edge gradient information (Figure B-3). The gradient is determined by,

$$\Phi = \tan^{-1}({g_y}/{g_x})$$
$$\lambda = \Phi - {\pi}/{2} \qquad (6)$$
$${dy}/{dx} = \tan(\lambda)$$

where

$g_x$ = edge gradient in the x direction

$g_y$ = edge gradient in the y direction

$\Phi$= orientation of the edge normal vector

$\lambda$ = orientation of the edge direction vector

dy/dx = slope of the line tangent to the parabola at a point, **p.**

Each pixel in the edge image over a certain threshold votes for all the possible parabolas (defined by the parameters) constrained by the analytical function in equation (5). Parabolas are then "peaks" in the parameter space. We use a simple threshold to calculate a peak. For each possible parabola detected, the algorithm must also maintain the calculated focal length, ***a***, for the parabola "peak" where "peak" is defined as the edge pixel furthest from the parabola vertex. Again, the focal length determines how narrow or wide the detected parabola is.

We add a post-processing step to remove false positives and consolidate several similar detected parabolas. The false positive parabolas are the result of discretetization errors (assumption that each edge pixel center is the center of the pixel) and localization errors. Localization errors result either when the parameters of a curve do not receive votes from edge pixels that are a part of the curve or because the discretization errors causes a single bin in the vote/accumulator array to receive a large number of votes from edge pixels that cannot lie on the same curve. The assumption is that the bin size is sufficient to receive the votes for parabolas of interest but small enough not to receive votes from false positives (Olson, 1999). Because of this induced error, we post process the parabolas by "walking the parabola" a few pixels in either direction from the vertex to make sure there are a sufficient number of edge pixels on the parabola to classify it as a curve.

The general algorithm follows:

1. Convert the source image to a grayscale image
2. Use an edge detector (we used a rotation invariant kernel mask) to find the set of edge pixels in the image along with their associated gradient information in both the x and y directions ($g_x$, $g_y$).
3. For each edge pixel
    a. Calculate the edge slope, dy/dx, according to equation (6)
    b. Iterate through some fixed rotation angle, $\Theta$, from 0 to $\pi$ by some step size, s
        1. Iterate through each possible $x_0$ (x coordinate of possible vertices)
            a. Calculate $y_0$ according to equation (4)

b. If ($x_0$ ,$y_0$) is an edge
1. Transform to canonical form (zero degrees of rotation from the x-axis with the vertex at the origin).
2. Determine actual angle (angle may be $\Theta + \pi$)
3. Determine focal length, **a**, based on equation (1)
4. If the focal length is within constraints (i.e. not a straight line and the parabola has an edge pixel opposite the current edge pixel), then cast a vote for the curve parameters ($x_0$ ,$y_0$, $\Theta$). If the current pixel is the "peak" pixel, then record its focal length for these parameters.

4. Post process to remove any false positives.

Parameters:

$a_{min}$ = 5
$a_{max}$ = 40
vote threshold = 340
step size, s = 5 degrees (0.087 radians)

Although the representation used for detecting curves is "depictive," the Hough transform is a "sentential" algorithm. That is, the algorithm detects curves by fitting the representation to analytical algebraic equations. One could argue that it is "biologically" inspired since it is highly parallelizable with a short dependency tree. We could parallelize an iteration through the set of edge pixels and angles of rotation, as each edge pixel vote is independent of the others. The current implemented algorithm is significantly slow, $O(e^{\pi/s}{}^{m})$, where *e* is the number of edge pixels in the image, *s* is the angle step size, and *m* the width of the image in pixels. We may obtain speedup using randomization and decomposition as described in (Olson, 1999) and implementing with multiple processors.

## B.2 Depictive Manipulations

Some of the *VBManipulator*'s (Figure 6-14) processing units are implemented as a *pixel-level rewrite* system (Furnas, 1990, 1991; Furnas et al., 2000; Yamamoto, 1996). Unlike sentential algebraic algorithms, such as a Gaussian filter or the Hough transform that take advantage of the geometric properties of the depiction, this type of processing takes advantage of the topological structure and color of a depictive representation. This

section will discuss our specific implementation details of this type of processing and the rewrite rules used in the experiments.

A pixel-level rewrite system has a set of *depictive rules* and a shared image. Similar to a production system, the depictive rules have a left-hand side (LHS) and a right-hand side (RHS) but rather than predicate symbols, the LHS conditions and RHS actions are visual depictive representations. The color of each LHS pixel and their spatial arrangement, or shape, determines a match rather than the syntactic structure of the symbols. Figure B-4 shows an example of two depictive rules. The top rule is a 2x1 rule stating, "if there is a black pixel adjacent to a gray pixel, then change the gray pixel to a white pixel." Similarly, the bottom rule is a 2x2 rule that says, "if there is a black pixel diagonally adjacent to a gray pixel, then change the gray pixel to a white pixel." The asterisks represent wildcard values where the processing ignores those pixel values in the determination of a match. Note that "color" simply implies that the pixel has an integer value (**F**-value). In addition to the orientation shown, a rule may specify that the processing also check for matches at 90, 180, and 270 degrees or for reflection. For example, the second rule, rotated 90 degrees counterclockwise matches a pattern in the image where there is a black pixel in the lower right corner and a gray pixel in the upper left corner. The RHS action changes the upper left pixel to white.

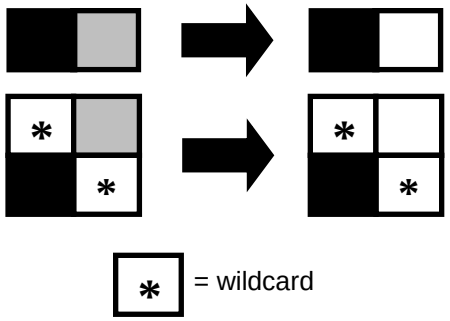



**Figure B-4: Pixel-level Rewrite Rules**

The processing iterates over the image, searching for a match of any rule's LHS pattern. If there is a match, the RHS action rewrites the appropriate pixel(s). Processing terminates when there are no rules matching a pattern in the image. To achieve control, each rule has a priority associated with it so if there are multiple matches in a particular pixel neighborhood, then the rule with the highest priority fires. Although the matching

and modifications are local in nature and extend no further than a 3x3 neighborhood in our particular algorithms, the overall effects of the manipulations have global consequences. The pixel rewrite model used in this first implementation has some details that might not be biologically reasonable (e.g., global conflict resolution with only one rewrite proceeding at a time), its essential nature of computation by iterated local transformations does not seem at all beyond the realm of neuro-biological possibility.

In order to integrate the pixel-level rewrite rules with the Soar+SVI architecture, we made the following three extensions. Yamamoto (1996) also investigated some of these extensions. First, one or more *depictive rules* are encoded in Soar as operator elaborations[30] (Figure B-5). When Soar selects the operator for application, the depictive rules are added to Soar's output-link. The VBManipulator receives the rules, and specialized processing units interpret and execute the matching and firing of rules. As shown in Figure B-5, each depictive rule has a name (for debugging), a priority, number of pixels (to indicate if it is a 1x1, 1x2, 2x2, or 3x3 rule), and any other rotation angles to check for a match (90, 180, 270). Rules may also have reflections associated with them although we did not use them in any of our tasks.

The second extension we made is to distinguish processing between three types of rules. Each rule is a member of a *rule-set* where all rules in a rule-set are constrained to be of the same *type*. The rule-set type signals to the VBManipulator the form of processing, and the rule-set *number* serves as a sequencing method (i.e. process rule-set 0, then 1, then 2, etc). We define three types of rule-sets: *Threshold*, *Pattern*, and *Mark*. Threshold rule-sets are rules with either a 1x1 or a 1x2 image on both the LHS and a 1x1 image (i.e. 1 pixel) on the RHS. The VBManipulator processes these rules by making one pass through the image and "thresholding" each pixel based on the LHS value where a 1x1 LHS signals an exact match and a 2x1 LHS image indicates a minimum and maximum range of values.

This functionality is different from a pixel-rewrite system in that rather than specifying a rule for each possible pixel value requiring change, a 2x1 LHS in a threshold rule specifies a range of pixel values where the values are on an ordinal rather than a nominal scale. We found this functionality useful in the Scout domain for marking known

[30]An operator elaboration is a type of Soar production or rule.

obstacles such as buildings and "no-go" terrain that could be defined roughly as a range of pixel values in the original image. We think a similar type of rule-set based on location may also be useful (e.g., we know the enemy is maneuvering through locations in a 9x9 region centered on location (x, y) even though it is considered impassable). However, we have not implemented this type of rule.

If there is an imagery operator proposed to transform a visual buffer layer with a set of depictive rules and there are *distance-field-flood color layers* then elaborate the operator as follows:

```
rule:
        name: <rule-name>                   # for debugging
        rule-set: <rule-set-number>         # 0 – N
        type: <rule-type>                   # Threshold,Pattern,Mark
        priority: <rule-priority>           # priority within this rule-set
        number-of-pixels: 1,3,4,9           # 1x1, 1x2, 2x2, 3x3
        rotate: 90 180 270                  # rotations to check for match
        lhs:                                 # Left-hand side (lhs)
            pixel:                           # one for each pixel (2, 4, or 9)
                number: 0 - 8               # pixel number based on its location in the 1x1, 2x1,
                                            # 2x2, or 3x3 LHS image
                type: exact min-value max-value wildcard # type of match
                vector:
                   red:   <red-value>       # 0-255
                   green: <green-value>     # 0-255
                   blue: <blue-value>        # 0-255
        rhs:                                # Right-hand side (rhs)
           similar to lhs
```

**Legend:**

| | |
|---|---|
| # | Comment |
| <var-value> | Variable |

**Figure B-5: Example Soar Operator Rule (in English format) for Depictive Manipulations**

The second type of rule-set is the *pattern.* The pattern rule-set processing is similar to the pixel-rewrite system (Figure B-4). The general algorithm for processing the rules is:[31]

```
while (!quiescence)
        quiescence = true
        for each pixel in the image
           get the 3x3 neighborhood
           check for a match in order of rule priority
           if match then quiescence = false
```

[31]Much more efficient algorithms are possible (Furnas & Qu, 2002), but their implementation is rather complex and not the main thrust of this research. Their increased efficiency would only strengthen the claims of this research.

The algorithm asymptotic run time is O(*nr*) where *n* is the number of pixels in the image and *r* is the number of rules. If *n* or *r* is large, then we pay a computational cost. We currently have no constraints imposed on the number of rules (*r*). However, by focusing the computations on a subset of the image, we can keep *n* small so our algorithm is effectively linear in the number of rules.

One way to keep the image size small is a consideration of our third type of rule processing, the *mark*. Again, processing is similar to the pixel-level rewrites in that the local neighborhood of the pixel determines activation. However, rather than processing the entire image, the processing starts at a location specified in the rule-set header (not shown in Figure B-5) and only considers the local 3x3 neighborhood of the current pixel. The rule-set may specify location as a pixel location or a visual object in the Object Map that is then projected onto its 2D pixel location. For this type of rule, the processing only considers the local 3x3 neighborhood of the current pixel rather than the entire image. The processing proceeds in a fashion similar to pixel rewrites in that if there is a match in the current pixel neighborhood, the pixel is marked according to the rule RHS. For mark rules, the RHS always specifies a modification of the center pixel. The next 3x3 neighborhood considered for matching is in the *direction* of the current match.

For example, in Figure B-6, if the current pixel is white and its diagonal pixel is gray (in any direction), then the current pixel is marked orange, and the processing shifts to the gray pixel. If the rule has a match in more than one orientation, then the processing records any other matching pixel locations and pushes them on a stack. After processing in the chosen direction is exhausted, the pixel locations on the top of the stack are iteratively popped and, if not already marked,[32] they are processed. For these types of rules, their depictive specification automatically includes all rotational directions (i.e. 90, 180, 270), and they are constrained to a 2x1 or 2x2 diagonal rule.

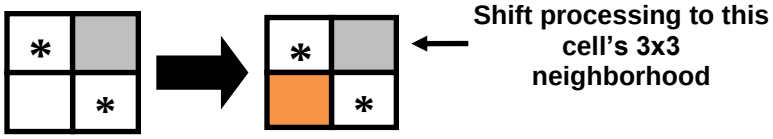



**Figure B-6: Mark Type Rule**

[32]This situation may occur when processing a closed region.

A mark rule-set is also distinguished from a pattern rule in that in addition to performing the pixel rewrites, the VBManipulator creates a symbolic shape object and adds it to the current scene's *Visual Feature Set*[33] in VS-STM. The shape object includes an emergent-id, the marking color, and the set of points defining the shape. We found this rule-set type useful for marking the hypothesized enemy paths in the Scout domain and propose that this form of processing is useful for marking salient objects by the Saliency Inspector during bottom-up visual processing.

The final extension we made to the pixel-rewrite system is to create an *attention window* in order to keep the image size (*n)* small. Since the depictive representation in the Visual Buffer contains more information than can be processed, an attention window focuses the processing effort (Kosslyn, Thompson, & Ganis, 2006). In our implementation, the attention window is a fixed, m x m region of the image, where m is a factor of two. The size of the attention window and its shift direction is task knowledge transmitted from Soar to the VBManipulator every decision cycle in which the manipulation is active.

For example, in the Scout Domain the attention window is set to 64x64 pixels. During the distance field flood, the attention window starts centered at the key terrain. After the flood completes in the current attention window, a Soar operator tells the VBManipulator to shift the window towards the enemy map-icon. Since the visual object of the enemy map-icon is in VS-STM, the VBManipulator simply looks up the visual-object's scene graph node, determines its location, and then projects that 3D location to a 2D pixel location on the image. It then "shifts" the attention window a fixed distance in a straight-line towards the provided location based on the attention window dimensions and a shift factor sent from Soar. For example, if the attention window is 64 x 64 and the shift factor is 0.25, the shift is 64 * 0.25 = 16 pixels towards the given location. Note that this is an internal shift of the cognitive focus. The agent's "head" is not turning. We use the rectangle shape for simplicity and its amenability to the rest of Soar (of importance because of the strategic role of attention allocation). Furnas and Qu (2003) have also

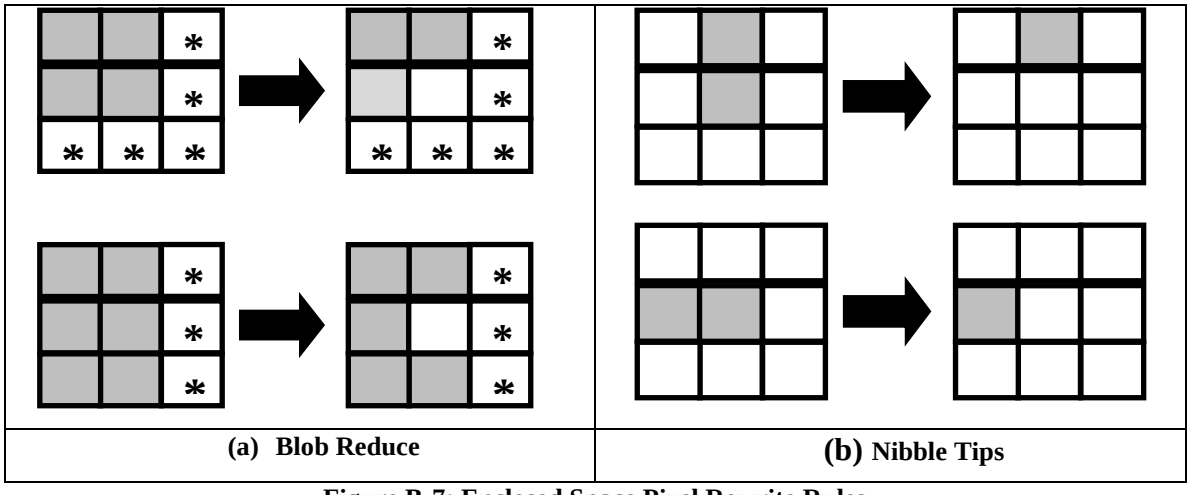
[33]If the marked shape is a subset of a specific object (e.g. the enclosed space of the letter A), then it is added to the Visual Feature Set of the corresponding Visual Object in VS-STM.

explored the notion of an attention window but where the processing is restricted to arbitrarily shaped regions rather than a fixed rectangular region. An interesting extension would be to explore using this more depictive (rather than spatial) notion of attention control.

As an example of how an agent uses pixel-level rewrites to infer new visual and spatial properties, consider the following rules from the Alphabet experiment. To find enclosed spaces in a letter, there are two basic pattern-matching rules (Furnas, 1990, 1991; Furnas & Qu, 2003). *Blob reduce* modifies a 2x2, 2x3, or 3x2 blob by removing the center pixel (Figure B-7a). *Nibble tips* reduces columns or rows of pixels in the image by removing the end of the column or row segment (Figure B-7b). Using these two basic algorithms, the general algorithm for finding enclosed spaces is as follows (sequenced using rule priorities):

1. Reduce 2x2 blobs
2. Reduce 2x3 or 3x2 blobs
3. Nibble Tips
4. If there is a remaining shape then there is an enclosed space.

Figure B-8 shows the letter B before and after processing using these manipulations.



**(a) Blob Reduce** **(b) Nibble Tips**

**Figure B-7: Enclosed Space Pixel Rewrite Rules**

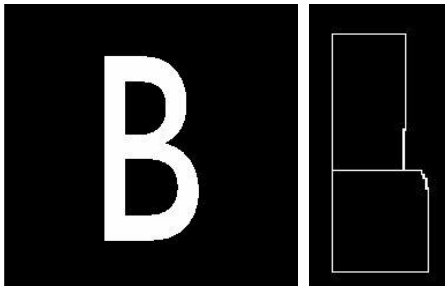
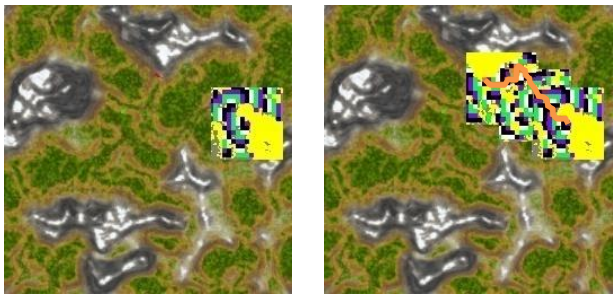


**Figure B-8: The Letter B Before and After Pixel Rewrites**

As another example, consider a task from the scout domain. The agent may analyze its team's position by imagining a hypothesized path from the current location (source) of each enemy vehicle to a key terrain location (sink). After the imagined path(s) are marked for each enemy/key-terrain pair, the agent imagines the team's view to determine if they have adequate coverage of the paths. The analysis should take into account the agent's knowledge about the surrounding terrain and known obstacles. A possible solution is the following:

1. Mark all known obstacles and "slow-go" terrain on the map with a color (yellow) by applying a set of known threshold values. Mark all other pixels gray.
2. Grow an iso-distance contour field avoiding any previously marked barriers (Figure B-9a).
3. Walk the contour field from source to sink, marking the path along the way (Figure B-9b).



**(a) Distance field flood** **(b) Mark Path**

**Figure B-9: Example Use of Pixel Rewrites from Scout Domain**

This task knowledge can be encoded as depictive rules using the following three rule-sets (Figure B-10). The first rule-set, shown in Figure B-10a in order of rule priority,

is a set of threshold rules to mark the known obstacles, steep terrain, and open terrain. For example, the agent marks known obstacles in green on the original depiction so the top rule changes those values to yellow, signaling an obstacle or barrier. Likewise, the middle two rules mark the steep and open terrain as yellow obstacles based on pixel ranges determined from an off-line analysis. The last threshold rule signals that any pixel in the image not meeting the above criteria should be marked gray to facilitate building the distance field flood.

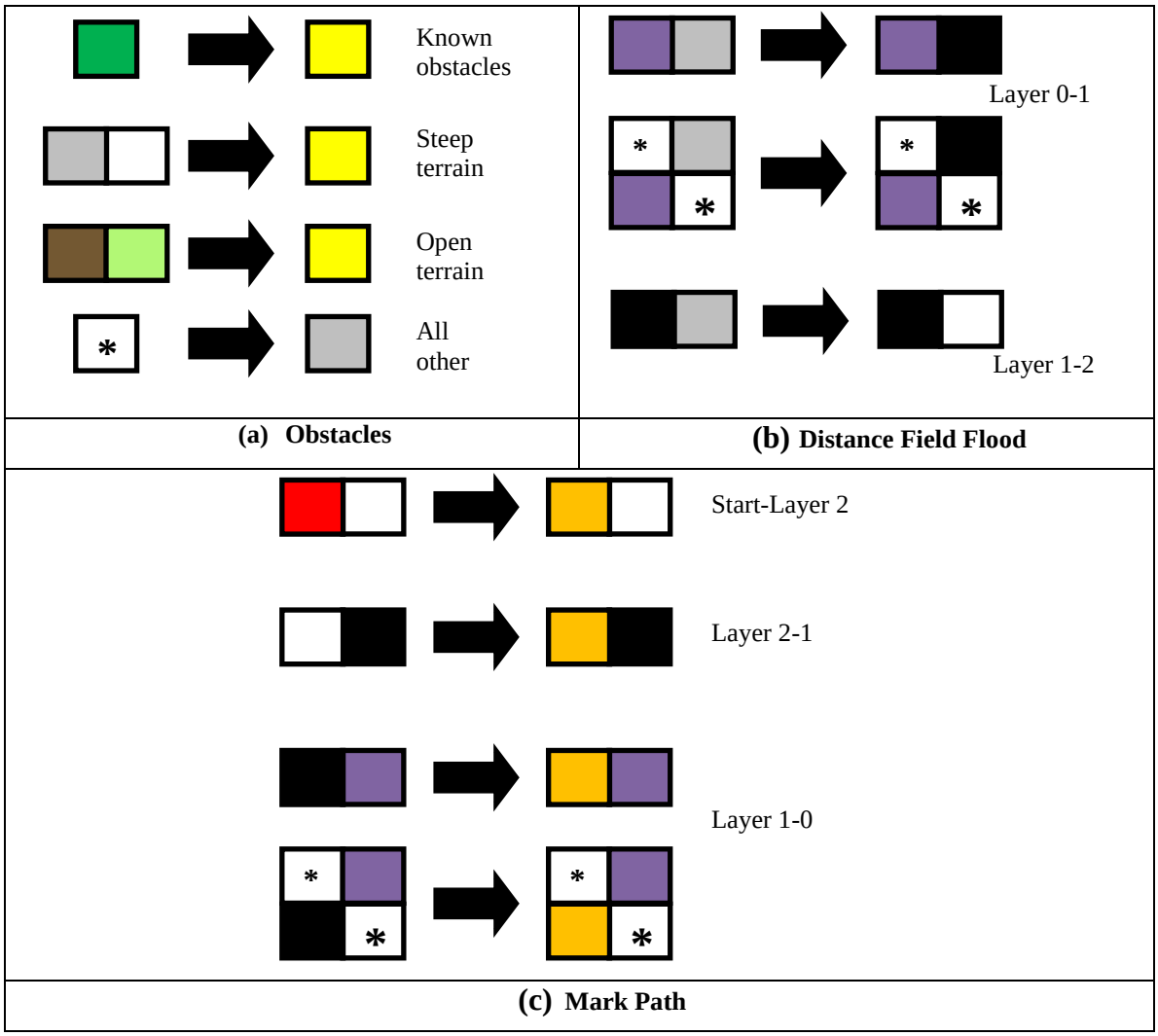



**Figure B-10: Depictive Rules for Scout Domain**

The second rule-set, shown in Figure B-10b, are a subset of the rules used to create a temporary iso-contour distance field flood starting from the "purple" sink (i.e. key terrain). These rules create alternating layers of four colors: purple, black, white,

green (not shown), purple, black, etc around the sink one pixel layer at a time (Furnas et al., 2000). Figure B-10b shows the rule for layer 0 (purple) to layer 2 (white). The even-to-odd layers match both orthogonal and diagonal pixels while the odd-to-even layers match only orthogonal pixels. Four colors are sufficient to preserve the topological shape around the yellow obstacles. The processing fills in the gray background pixels with the contours. The end effect is a downhill field gradient that has both distance and directional information that serves as an attractor for the subsequent, path-marking phase.

The final rule-set, shown in Figure B-10c, are rules that mark the path starting from the red source (i.e. enemy map icon) to the purple sink (i.e. key terrain). The rules take advantage of the direction and distance information of the iso-contor distance field to find the shortest path from source to sink that avoids the yellow obstacles. Figure B-10c illustrates the first few rules that fire, marking the path orange (assuming the red source is initially adjacent to a white distance field layer—other rules are required for the remaining initial possibilities). Note that once the top rule fires, moving the processing from the source (i.e. red) to a distance field color (purple, black, white, or green), the rules are simply the opposite from the rules in Figure B-10b used to create iso-contour field. Each layer "attracts" the path from the previous layer while avoiding obstacles. Orthogonal directions are preferred (i.e. have a higher priority) than diagonal directions.

There is a default rule (not shown in the figure) that fires when there is not a match on the current pixel, and the current pixel is not purple (i.e. not the sink). In the Scout domain, this situation may occur at the start or sometimes on the border of an attention window when the current pixel being processed is yellow, or an obstacle. For example, the enemy map-icon may be located on a piece of terrain that the agent thought was "no-go" terrain. In this situation, the default rule behavior is to move towards the sink based on the annotated default direction preference of the rule.

# Appendix C

# Software Engineering and Implementation

The following appendix describes the software design and implementation in enough detail to give the reader an understanding of the major software components, classes and their associations, and the symbolic representations in Soar's working (i.e. short-term) memory. We will start by listing the software libraries and their dependencies. Then, similar to the architectural discussion in Chapter 6, we will discuss SVI memories and processes for spatial and visual imagery processing (Figure 6-1) using the Unified Modeling Language (UML) diagram notation. As outlined in the architectural view, connections between components imply both data and control constraints. We model these constraints as UML associations (aggregate, composition, inheritance). The last section describes the relevant Soar structures.

For performance, functionality, and usability reasons, the system is a layered architecture with the mathematical functionality and image processing written in the C++ programming language and the software interfacing SVI, Soar, and its debugging tool, the SoarJavaDebugger, written in the Java programming language. We use the following open source software packages: CImg (Tschumperlé 2008), OpenGL (Shreiner et al., 2006) and the corresponding LWJGL ("Lightweight Java game library (LWJGL)", 2008), Soar ("Soar", 2008), SWT ("The standard widget toolkit (SWT)", 2007), SWIG (Beazley et al., 2002), and Wild Magic (Eberly, 2005). Wild Magic provides the basic mathematical package and scene graph support, CImg has the image data structure and

algorithms, OpenGL is a software interface to graphics hardware, and SWT is a toolkit for graphical user interfaces. We use the Soar Markup Language (SML) to interface Soar with SVI and SWIG to generate the wrapper code for bridging C++ and Java code. Figure C-1 and Figure C-2 illustrate the major software components and dependencies.

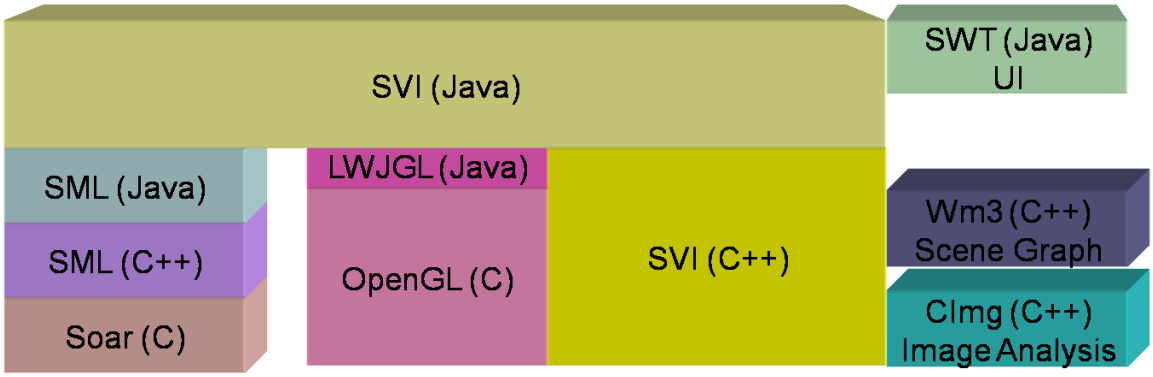



**Figure C-1: SVI Libraries**

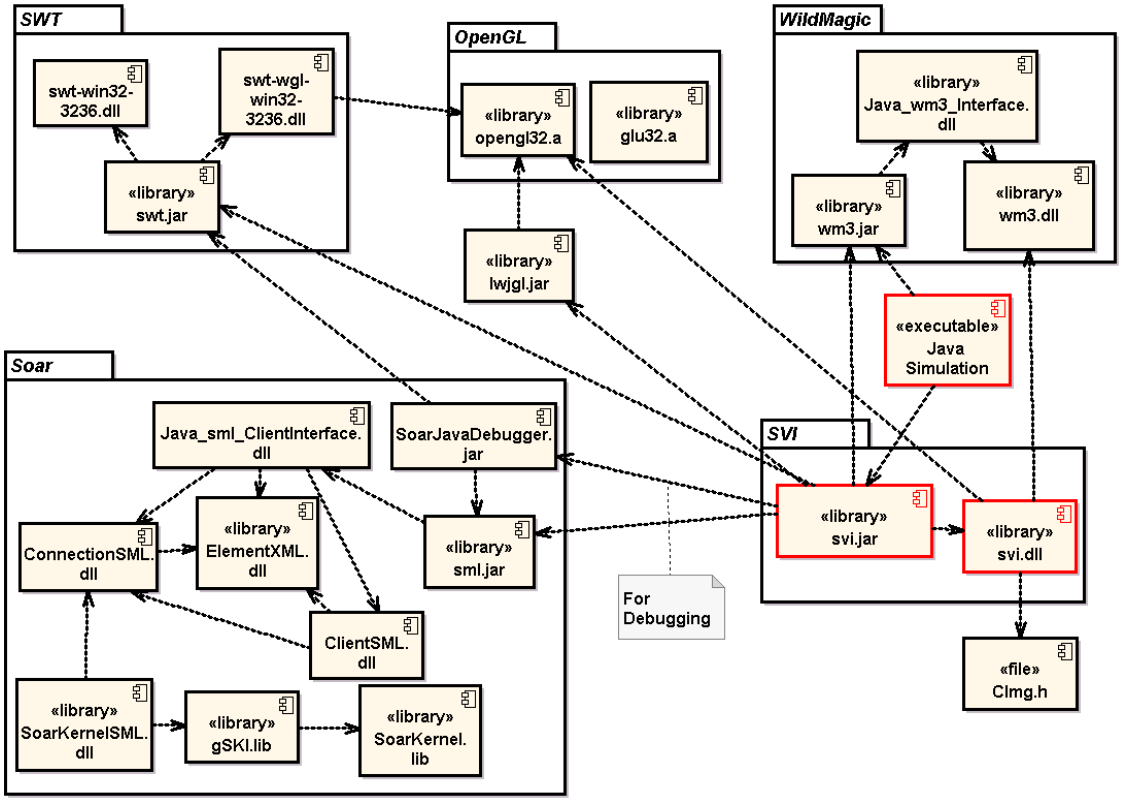



**Figure C-2: Component Dependencies**

## C.1 Class Diagrams

The *VisualObject* is a basic class in the system. It is composed of zero or more VisualObjects, *VisualFeatures* and *SpatialProperties* (Figure C-3). Along with its instance-id, and if "recognized," visual-id, a VisualObject has a boolean flag indicating whether or not it is perceived or imagined. A VisualFeature may be a *Color* or a *Shape* and has a unique emergent-id. A SpatialProperty is an instance of *Direction*, *Distance*, *Orientation*, *Topology*, *Geometry*, or *Size* and has a relative-instance-id or –emergent-id and perhaps a base-instance-id or –emergent-id depending on the type (i.e. unary, binary, tertiary) of spatial property and the VisualObject(s) or Shape(s) defining it. Visual-Spatial short-term memory (Figure 6-6) is simply an instantiation of a VisualObject that represents the current scene. The *VisualBuffer*, *ObjectMap*, and *VisualLTM* share this VisualObject, effectively binding the "what" and "where" pathways.

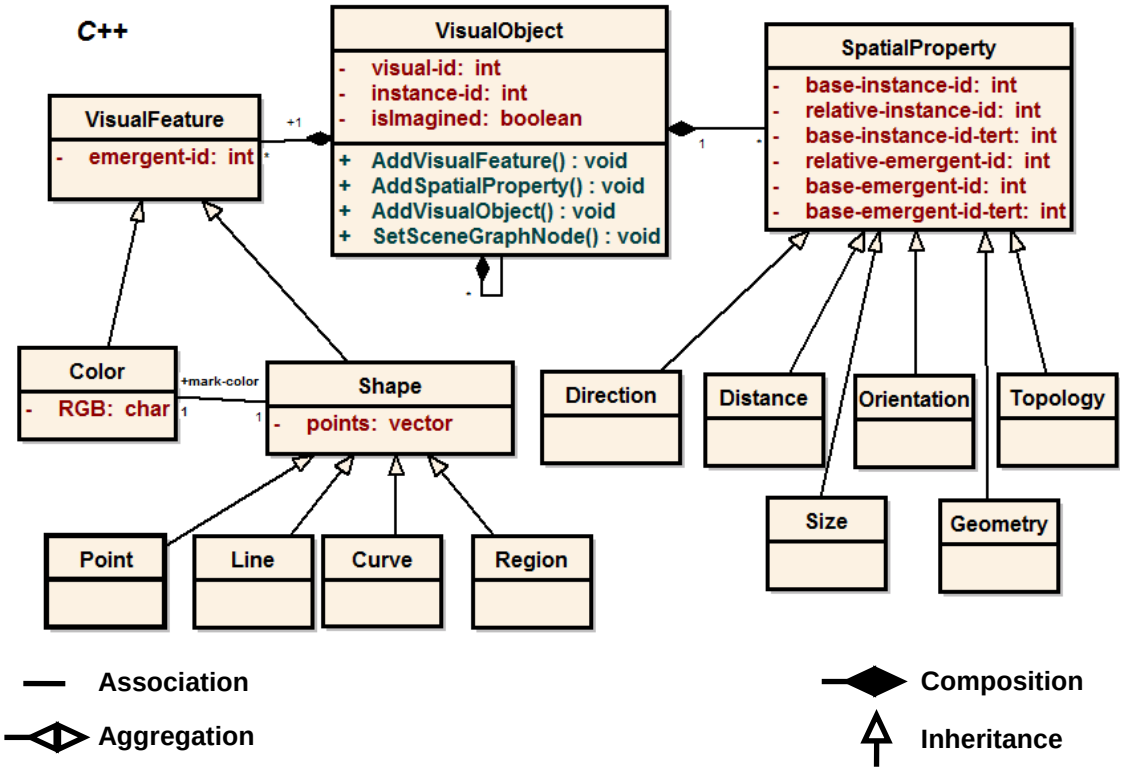



**Figure C-3: Visual Object Class Diagram**

Within the Java SVI component, a *SoarAgent* class serves as an object adapter for the SML *Agent* and *Kernel* classes (Figure C-4). A *SoarAgent* has zero or more

*ISoarComponents* ('I' is the Hungarian notation for "interface") that are either memories (i.e. *VisualLTM*) or processes (i.e. an *Inspector*). The SoarAgent class defines two inner classes, *InputLink* and *OutputLink*, each consisting of a collection *ISoarInputLink and ISoarOutputLink* objects. During initialization, each ISoarInputLink and ISoarOutputLink object in the system registers with the SoarAgent's respective InputLink and OutputLink objects for the commands it handles. During Soar's output phase, the OutputLink object parses Soar's *output-link* and, for each type of command, calls ISoarOutputLink object responsible for parsing and executing the specific symbolic structures. Likewise, prior to the input phase, the InputLink object calls each of the registered ISoarInputLink objects to create or modify symbolic structures on Soar's *input-link.*

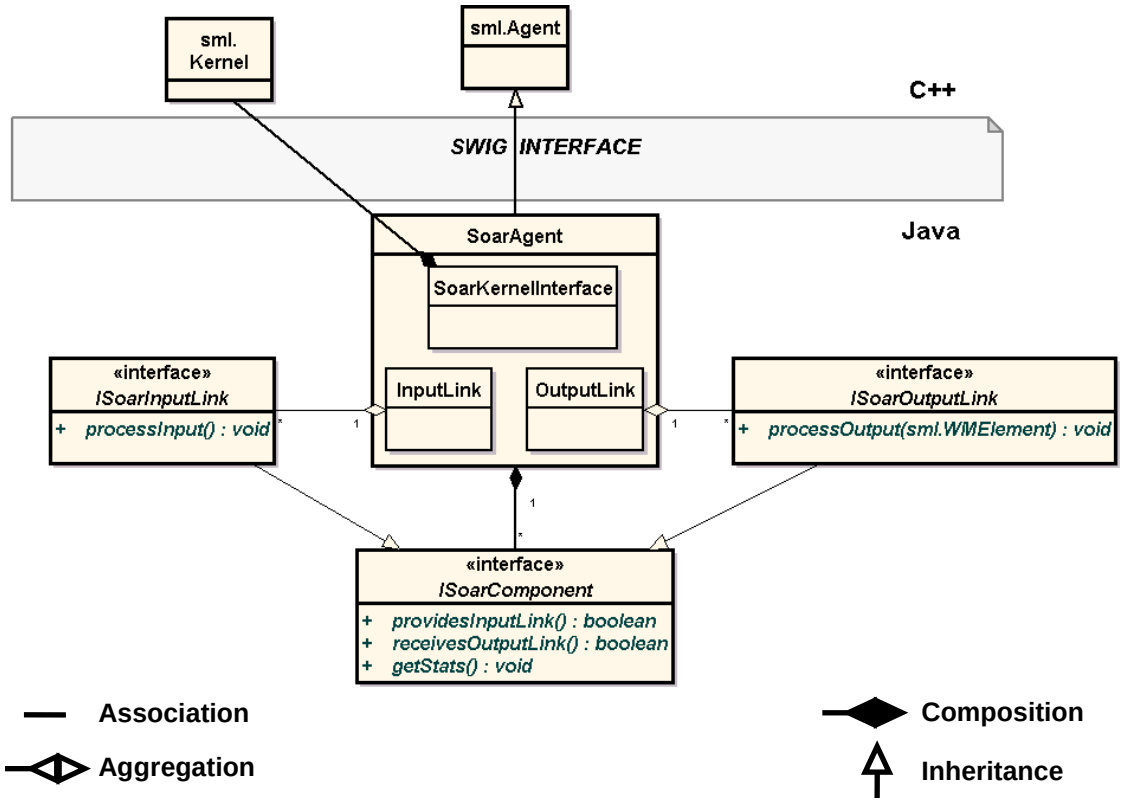



**Figure C-4: Soar Agent Class Diagram**

Every SVI component derives from either an *AgentMemory* or *AgentProcess* class that in turn derive from the *AgentComponent* class (top of Figure C-5). AgentComponent encapsulates basic information and behavior that all of the processes and memories require such as its type, its Soar agent, and functionality for gathering run-time statistics.

AgentMemory includes basic memory behavior, such as storing and retrieving, posting inspector results, and notifying listeners of those results. AgentProcess simply serves as a placeholder to distinguish between the two types of components using run-time type information.

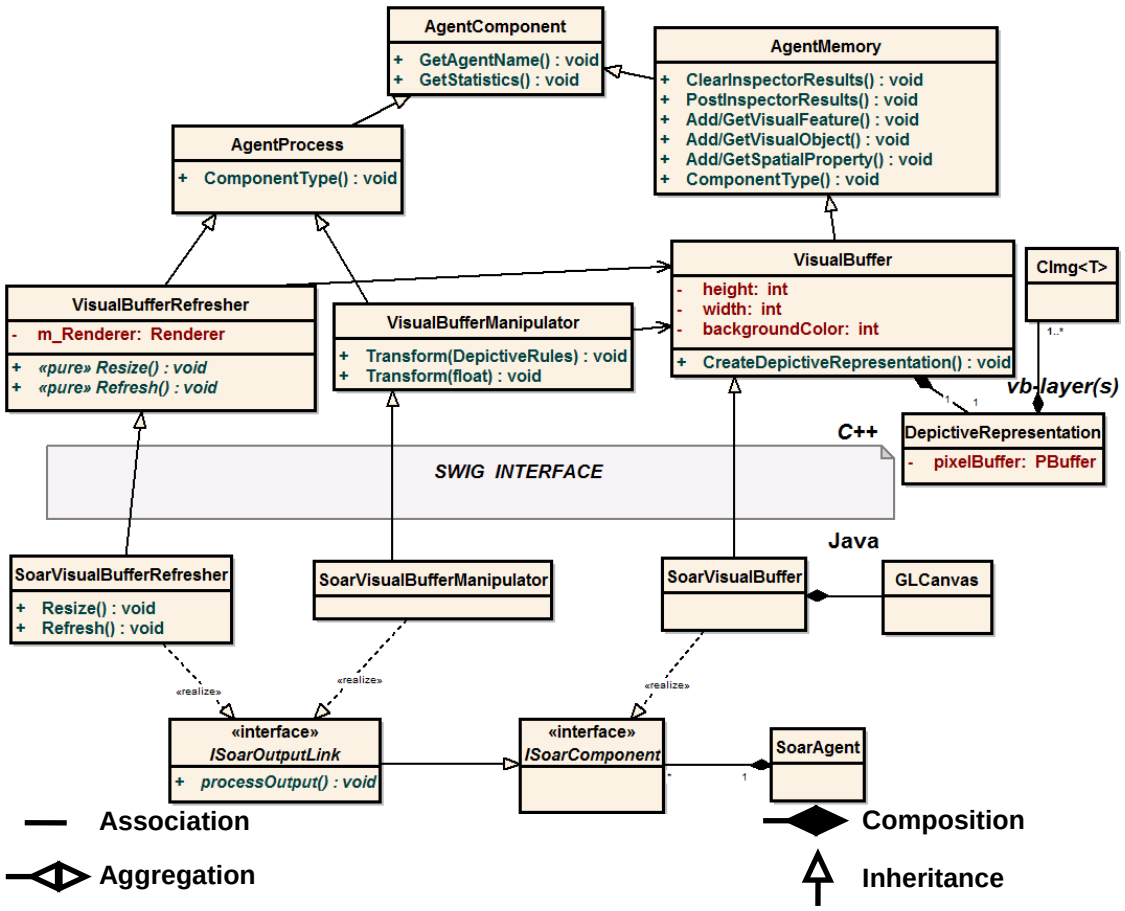



**Figure C-5: Visual Buffer Class Diagram**

The *VisualBuffer* derives from AgentMemory and contains attributes for its height, width, and background color (Figure C-5). It encapsulates the depictive representation that the *VisualBufferRefresher* creates by rendering the scene to a display canvas (for debugging). The VisualBufferRefresher reads the pixels from the graphic processing unit (GPU) pixel buffer into a CImg (Tschumperlé 2008) data structure by calling the VisualBuffer's *CreateDepictiveRepresentation* function. For efficiency and functionality purposes, the pixel buffer is read only when the agent issues a *generate* command. Otherwise, the VisualBufferRefresher renders the image to the display canvas only. The image read from the GPU is the base visual-buffer layer (*vb-layer* 0). The

*VisualBufferManipulator* and inspection process create other images from the base image as required and store them temporarily in the VisualBuffer as a set of CImg structures.

All agent components written in C++ for SVI are extended in Java and have a corresponding Soar<ComponentName> class (e.g. *SoarVisualBuffer*). These classes serve three purposes. First, they implement the *ISoarComponent* (Soar Component Interface) providing them with the interface to register as components of the *SoarAgent* object (Figure C-4 and Figure C-5). This requirement is for ownership purposes so that the *SoarAgent* object maintains responsibility for the memory allocation of its components. Second, some of the Soar+SVI classes interface with Soar's input and/or output links. For example, the *SoarVisualBufferRefresher* and *SoarVisualBufferManipulator* implement the ISoarOutputLink interface. This standard interface provides each component with the functionality to receive commands (i.e. generate, transform) from Soar. Third, the Soar+SVI classes may provide additional functionality specific to their Java implementation. For example, the *SoarVisualBuffer* implements an SWT OpenGL drawing canvas (*GLCanvas*) that is specific to Java facilitating the integration of SVI with Soar's debugging tool, the SoarJavaDebugger (Figure C-6). The *SoarVisualBufferRefresher* implements the necessary threading and Lightweight Java Graphics Library to draw a scene to the GLCanvas.

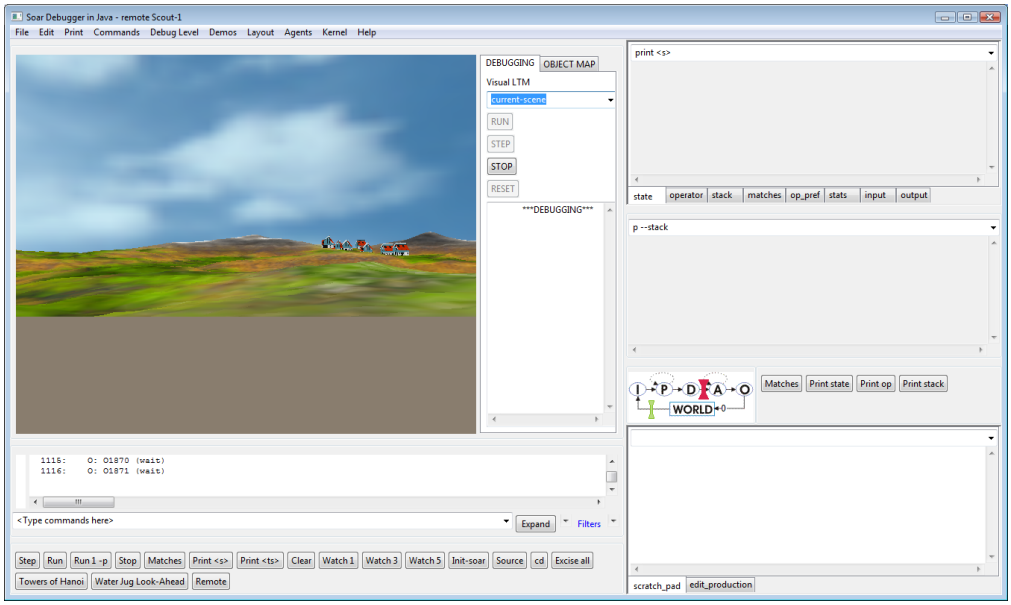



**Figure C-6: SoarJavaDebugger with SVI Interface**

*VisualLTM* (Figure C-7) is a hash table indexed by a *visual-id*. Each *VisualEntry* includes a visual-id, a scene graph representing the object, and an association with other entries containing the object's parts. To assist construction, the VisualEntry class stores basic statistics concerning each object's vertices (e.g. minimum and maximum vertices). VisualLTM also provides an interface to load and store scene graph objects from a file system. *SoarVisualLTM* extends VisualLTM so it can register as a Soar component.

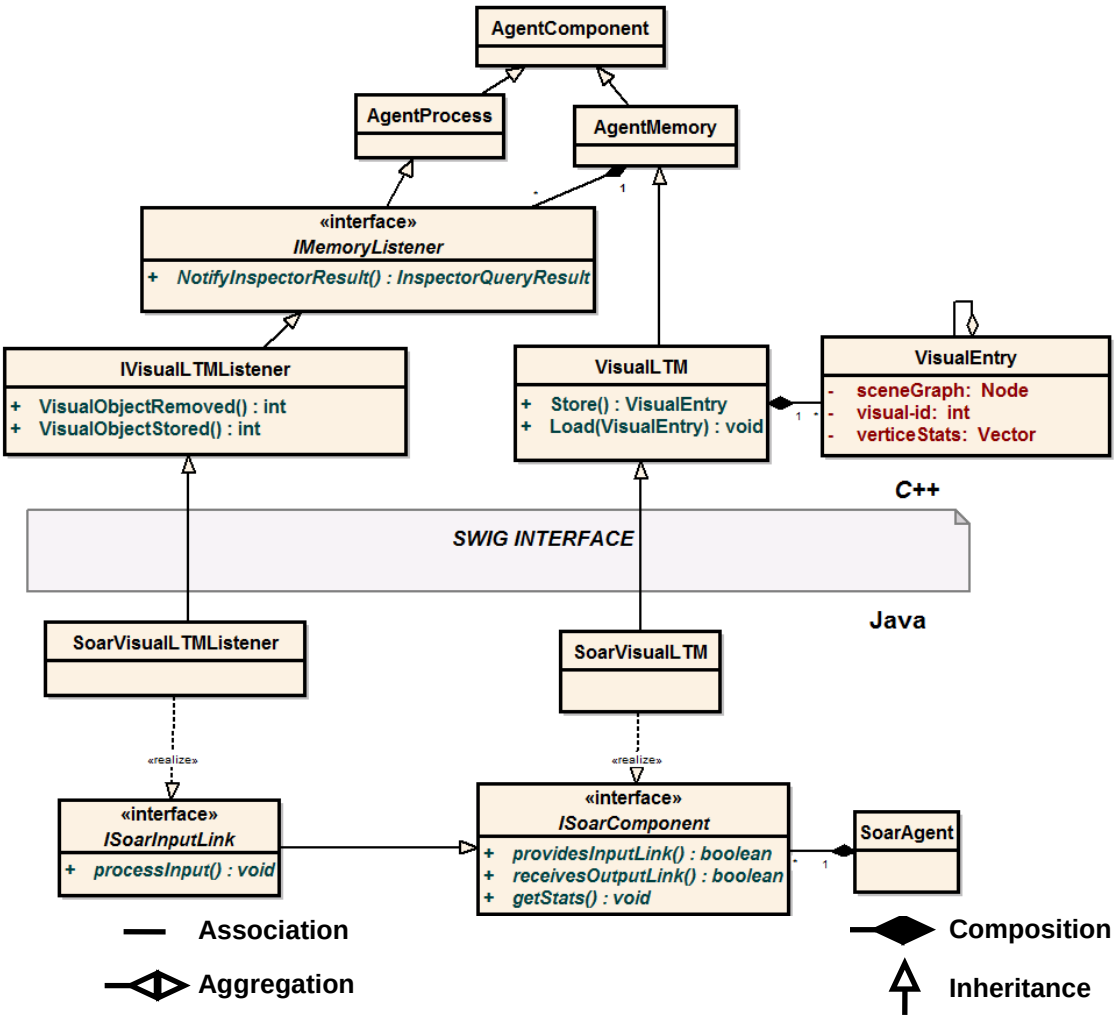



**Figure C-7: Visual Long-term Memory Class Diagram**

Every memory has one or more *IMemoryListener* interfaces associated with it. For VisuaLTM there is a C++ *IVisualLTMListener* and a corresponding Java *SoarVisualLTMListener* (Figure C-7). The SoarVisualLTMListener implements the ISoarInputLink interface, enabling it to communicate with Soar's input-link. After

recognition of a visual object, manipulation of the VisualBuffer, or inspection for visual features, the SoarVisualLTMListener receives a signal from the appropriate process. During the subsequent input phase, the SoarVisualLTMListener creates the appropriate symbolic structures on Soar's input-link from the information in VS-STM.

The *ObjectMap* and *IObjectMapListener* have a similar design to VisualLTM and IVisualLTMListener (Figure C-8). The ObjectMap has a *Node* representing the current scene graph and a *View* that represents the location and direction of the agent's viewpoint. The *ObjectMapConstructor* and *ObjectMapManipulator* have direct access to the ObjectMap so that they can construct and manipulate the scene graph or change the ObjectMap's viewpoint. Their corresponding Soar class definitions implement the ISoarOutputLink interface in order to receive commands (construct, transform) from Soar. The *SoarObjectMapListener* communicates spatial query results stored in VS-STM by creating symbolic structures on Soar's input-link.

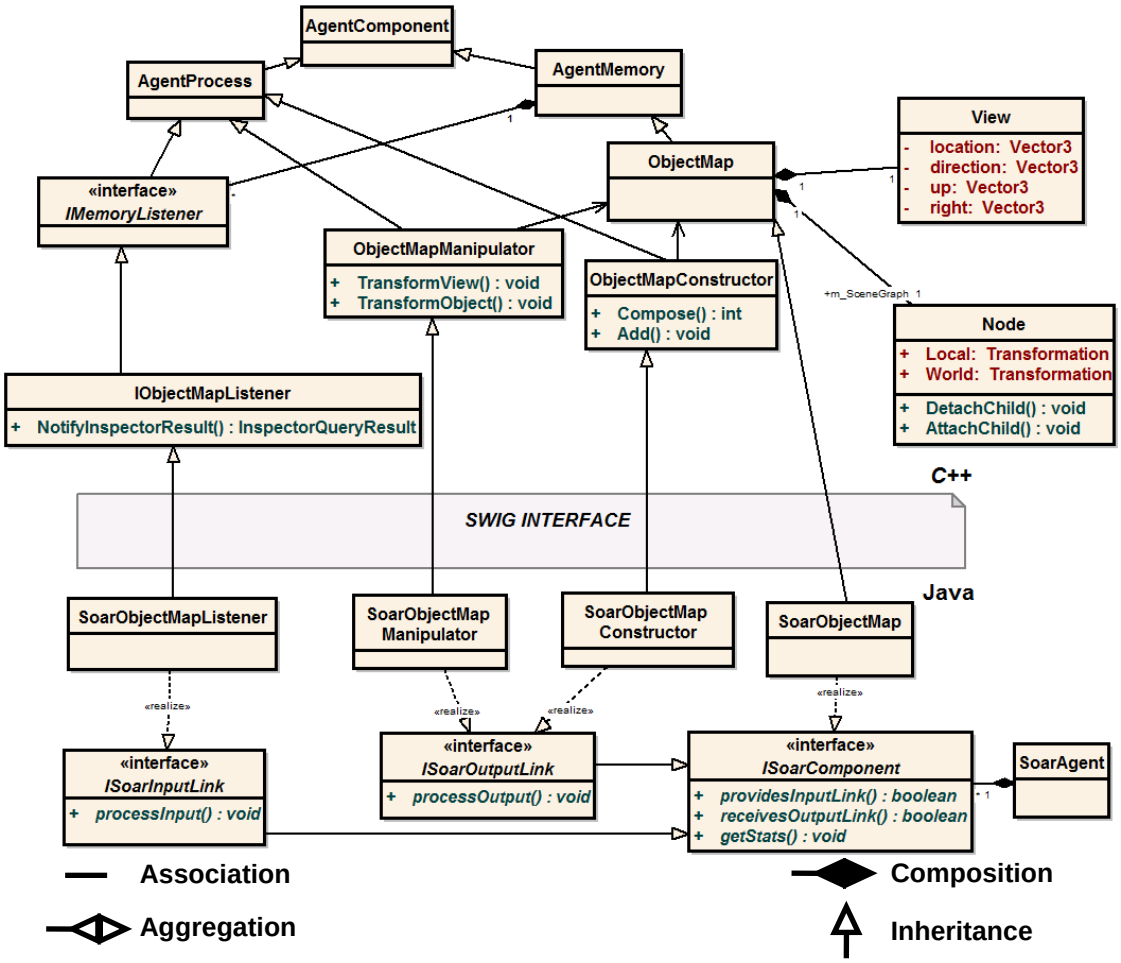



**Figure C-8: Object Map Class Diagram**

An *InspectorManager* object maintains a reference to all registered *VisualSpatialInspectors* in the system (Figure C-9). These processes inspect the VisualBuffer or ObjectMap in response to an automatic bottom-up query or top-down imagery inspection. There are three types of VisualSpatialInspectors. The first is a SaliencyInspector that is responsible for initially marking and attempting to recognize objects in the perceived scene. To facilitate recognition, the saliency inspector through the InspectorManager may call the other two types of inspectors, *VisualFeatureInspector* and *SpatialPropertyInspector*. Both of these classes have several derived, concrete classes implementing the algorithms for a specific type of visual or spatial property (i.e. line, enclosed spaces, direction, topology, geometry, etc.).

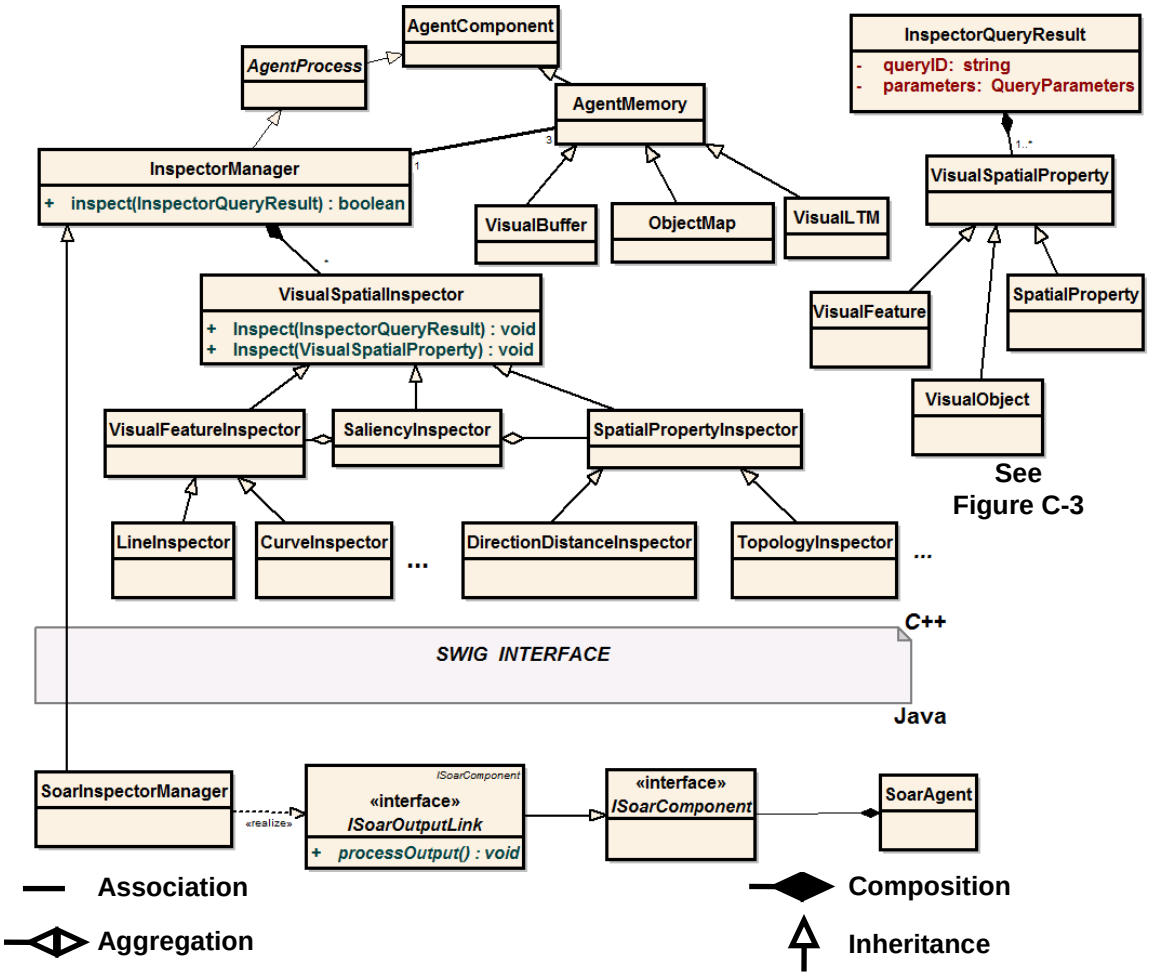



**Figure C-9: Inspector Class Diagram**

The *SoarInspectorManager* (bottom of Figure C-9) is responsible for parsing the inspect command from Soar and initiating the inspection process within SVI. When the SoarInspectorManager receives the command to inspect, it first creates an *InspectorQueryResult* object (top right of Figure C-9) that encodes the specific query and stores the collected results. The InspectorQueryResult has a *query-id* (a unique symbol generated by Soar prior to the inspection for tracking purposes), an optional set of query parameters (e.g. parameters for Hough transform, attention window size, query constraints) and one or more *VisualSpatialProperties*. The VisualSpatialProperty is an abstract base class for the VisualFeature, VisualObject, and SpatialProperty classes previously discussed (Figure C-3).

After marshalling the query by creating the InspectorQueryResult structure, the SoarInspectorManager calls the *inspect* function of the base class. Based on the run-time type information of the first visual-spatial property in the query, the InspectorManager determines and dispatches the appropriate VisualSpatialInspector by calling its corresponding *Inspect* function (Figure C-9). The InspectorManager knows what inspector to dispatch because during initialization, each inspector registers the type of visual-spatial property it is capable of processing.

The first inspector called during a query becomes the lead inspector and is responsible for posting any results to VS-STM. If during the inspection, the lead inspector comes across a visual-spatial property that it cannot handle, it dispatches the appropriate inspector through the InspectorManager by calling the second *Inspect* function shown in the VisualSpatialInspector class (Figure C-9). This function signals to the called inspector to perform the inspection, store the results in the provided VisualSpatialProperty object, and return to the caller without posting results. For example, if the first visual-spatial property in a query is for the direction between two visual objects and after finishing the inspection, the *DirectionDistanceInspector* comes across a request for the topological relationship of the visual objects, it accesses the *TopologyInspector* via the InspectorManager and dispatches it. When the TopologyInspector finishes, it stores its results and returns control to the DirectionDistanceInspector. The DirectionDistanceInspector consolidates the results in

the original InspectorQueryResult structure, stores it in VS-STM, and notifies the appropriate memory listener that there are results waiting processing.

## C.2 Soar Symbolic Structures

Now that we have discussed the SVI classes, we can explain within this context how spatial and visual imagery processing works from Soar's perspective. We will discuss the symbolic structures from the point of view of someone writing an agent. Soar's short-term or working memory elements (WME) are a three-tuple (identifier, attribute, value). A value may be a primitive (integer, float, string) or another symbolic identifier. In most cases we only represent the attribute with a caret (e.g. ^attribute). If the attribute has sub-structure, we represent it either with a new-line and indentation or by connecting it with a dot to its parent attribute (e.g. ^attribute.sub-attribute). **Bold** entries are permanent architectural structures. Entries shown in *italics* are WMEs that the system creates (currently through productions) in response to input or a command. Normal text entries are working memory elements (WMEs) the agent creates with productions (i.e. task knowledge). Finally, entries in <angle brackets> are items suggesting the types of values that might augment the existing structure.

The imagery subsystem includes SVI and a set of Soar productions that initialize some working memory structures and facilitate communication with SVI through Soar's input- and output-links. We consider these productions and structures as "architectural" rather than knowledge. The imagery system uses Soar's subgoaling mechanism to implement the imagery processing specific to the input- and output-links. Soar's top-state working memory structure has a visual-spatial working memory (**vs-wmem**) attribute (Figure C-10a). This attribute has three architectural substructures: **imagery**, **visual-ltm**, and **visual-object-instances**. The agent issues imagery commands (construct, transform, generate, inspect) and receives results by augmenting the imagery attribute. We discuss this structure in more detail shortly. When a VLTM listener informs Soar that a visual object has been stored in VLTM,[34] an operator creates a *visual-object* structure under the **visual-ltm** attribute. Figure C-10b shows an example of a *visual-object* structure. Based

[34]In the current implementation, this input implies that the visual-object scene graph has been loaded from the file system.

on input from the VLTMListener, only the visual-id and has-a attributes are initially present in a **visual-ltm** *visual-object*. The name is optional and added by the agent to assist in identifying instances of the object and associating it with other symbolic structures (e.g. enemy tank). The agent adds explicit visual features and spatial properties as it acquires them (programmed or learned). Although we implemented the **visual-ltm** structure in working memory, we assume that it is better suited for one of Soar's long-term declarative memories (i.e. episodic or semantic).


**Figure C-10: Top-state Visual-Spatial Working Memory Structure**

The **visual-object-instances** (Figure C-10a) also store a set of *visual-object* structures. Unlike the structures augmenting the **visual-ltm** attribute, however, these symbols are short-lived instances of what the agent has recently perceived or imagined.

When the architecture first *recognizes* a visual object (i.e. an incoming *visual-object* structure on Soar's input-link with a visual-id and an instance-id), it builds a *visual-object* working memory structure under the **visual-object-instances**. The agent *identifies* the *visual-object* by associating it with its entity (e.g. the visual-object, "S" is associated with the letter 'S', the visual-object "enemy-tank" is associated with an enemy entity). The *visual-object* structure contains a visual-id, instance-id, and, if the agent "imagined" rather than perceived the object, an *is-imagined* attribute. The *has-a*, *visual-features*, and *spatial-properties* attributes are inherited from the *visual-object*'s corresponding visual-object structure with the same visual-id encoded in **visual-ltm**. The agent optionally adds a name.

Subsequent observations of a *visual-object* do not create a new *visual-object* structure but simply match the incoming instance-id with the stored instance-id. If there is not a match before building a new *visual-object* structure, the system determines if the perceived object is the same as an existing visual-object based on the visual-object's egocentric location and known velocity (semantic knowledge). If the incoming visual-object is within a certain radius (task knowledge) of an existing visual-object instance, the system assumes the incoming and existing visual object are the same and does not create a new structure. Any imagined visual-object is removed when the system switches from imagining to perceiving. Other visual-object instances, in theory (not implemented), decay over time and are removed from working memory when the current episode completes.

Figure C-10c-d also shows examples of a working memory structure for a single visual feature and spatial property that is similar to the SVI class diagram for those properties (Figure C-3). This information arrives on Soar's input-link from either the VisualLTMListener or ObjectMapListener and may augment a visual-object's visual-features or spatial-properties. The visual-feature attribute has an emergent-id and either a shape or a color sub-attribute. The color is expressed qualitatively (e.g. red, green, etc.) or quantitatively as a scalar or RGB vector. The shape attribute has a type (line, curve, etc.), number of points, a flag signaling whether or not it is closed (i.e. a region), and marking color. The spatial attribute describes the visual-object(s) or shape(s) involved with the

relative- and base-instance/emergent-ids. Except for topology and geometry, each spatial property may be described in qualitative or quantitative terms.

An agent invokes the imagery system by creating a command on the ^**imagery.command** structure (Figure C-11). The system augments each command with a unique command-id and after processing the request, creates a corresponding **result** structure with the same command-id. The purpose of the command-id is to facilitate the tracking of an agent's request with the result from the system.

An agent composes two visual-objects or adds a visual-object or shape to the current scene by augmenting the ^**imagery.command** structure with a compose or add command (Figure C-11a). The command includes a spatial structure (Figure C-10d) signaling to SVI how to configure the visual-object/shapes. For a compose command, both the relative- and base-visual-objects are from the set of visual-object structures in the **vltm** structure (Figure C-10a-b). The relative-visual-object is also from the **vltm** structure for an add command, but the base-visual-object is from the **visual-object-instance** structure as it is already an instance of the scene. If adding a shape as a first-class visual-object, then the agent must either specify the shape structure with its corresponding emergent-id so that SVI can access its vertices, or the agent must specify a set of points and their connections (not shown in Figure C-10c). After processing the command, the imagery system creates the *visual-object* structure on the ^**visual-object-instances** structure (Figure C-10a) and records the instantiated visual-object(s) on the ^**imagery.result.retrieved** structure (Figure C-11b). To manipulate the quantitative spatial or visual depictive representations, an agent creates a transform-om or -vb structure (Figure C-11b-d). To transform a visual object in the ObjectMap, the agent builds the spatial structure (Figure C-10d) with the relative-visual-object as the transforming entity. The agent may change its imagined viewpoint by specifying qualitative or quantitative information. To transform an image in the VisualBuffer, the agent creates the transform-vb structure with the information shown in Figure C-11e. The vb-layer is the image to transform. If using depictive rules to specify the manipulation then the attention window specifies its parameters (see Appendix B.2). If the manipulation is of type *mark*, then the system records a relative-shape structure on the ^**retrieved.***transform-vb* structure (Figure C-11f).

| | |
|---|---|
| ^**imagery.command**.compose/add<br>^*command-id* <unique-symbol><br>^spatial <spatial-struct> | ^**imagery.result**<br>^*retrieved.compose/add*<br>^*command-id <same-as-original>*<br>^*relative-visual-object/shape*<br>^*base-visual-object* # compose only |
| **(a) Construct Command** | **(b) Construct Result** |
| ^**imagery.command**.transform-om<br>^*command-id* <unique-symbol><br>^spatial <spatial-struct><br>^viewpoint<br>^qualitative front/top/side<br>^quantitative<br>^location<br>^direction<br>^up | ^**imagery.result**<br>^*retrieved.transform-om*<br>^*command-id* <same-as-original> |
| **(c) Transform ObjectMap Command** | **(d) Transform ObjectMap Result** |
| ^**imagery.command**.transform-vb<br>^*command-id* <unique-symbol><br>^vb-layer <image-to-transform><br>^attention-window<br>^width <number-of-pixels><br>^height <number-of-pixels><br>^shift-factor<between 0.0-1.0><br>^base-visual-object # start location<br>^relative-visual-object #stop location<br>^transform # image processing<br>^rotate<br>^scale<br>^rules #depictive manipulations<br>^rule # See Appendix B.2 | ^**imagery.result**<br>^*retrieved.transform-vb*<br>^*command-id <same-as-original>*<br>^*relative-shape* # if *mark* rule |
| **(e) Transform VisualBuffer Command** | **(f) Transform VisualBuffer Result** |
| ^**imagery.command**.generate<br>^*command-id* <unique-symbol><br>^visual-object-instances<br>^add-visual-object<br>^remove-visual-object | ^**imagery.result**<br>^*retrieved.generate*<br>^*command-id <same-as-original>*<br>^*vb-layer <generated-layer-num>* |
| **(h) Generate Command** | **(i) Generate Result** |
| ^**imagery.command**.inspect<br>^*command-id* <unique-symbol><br>^visual-object/vb-layer # visual-feature<br>^visual-feature<br>^visual-spatial | ^**imagery.result**<br>^*retrieved.inspect*<br>^*command-id <same-as-original>*<br>^*visual-feature*<br>^*visual-spatial* |
| **(j) Inspect Command** | **(k) Inspect Result** |

**Figure C-11: Imagery Command and Result Working Memory Structures**

An agent issues a generate command by augmenting the ^**imagery.command** with a generate attribute (Figure C-11h). The agent may create either one or more ^add-visual-object or ^remove-visual-object attributes to specify the visual-objects it wants generated in the visual depictive representation. After generating the image, the system augments the ^**result**.*retrieved.generate* structure with the generated vb-layer number (Figure C-11i) to support further reasoning such as manipulation of the image or queries for visual features.

Finally, the agent initiates the inspect command by creating the inspect structure as shown in (Figure C-11j). The ^visual-feature and ^visual-spatial structures are as illustrated in Figure C-11c-d. If the query is for a visual-feature, the agent must specify either the visual-object or vb-layer to inspect. SVI can determine a visual-object's associated vb-layer image from VS-STM (Figure 6-6). However, sometimes it is easier to specify the vb-layer if the agent just generated it and there are two or more visual-objects involved in the inspection (e.g. two lines).

After processing the inspect command, the imagery system creates the result structure (Figure C-11k). In the case of spatial queries with only a single, relative-visual-object specified in the original inspection command (e.g. what are all of the visual-objects left-of the fork), the ^**result**.*retrieved.inspect* structure contains a ^spatial attribute for each pair of visual-objects satisfying the constraint. For binary or tertiary spatial queries (e.g. is the plate right-of the fork), the system creates a single ^spatial structure if the assertion is true; otherwise, the attribute will be missing. If the agent desires quantitative information, then in the original command it specifies the attributes it desires. For example, for the query "What is the direction and distance between enemy-1 and the key-terrain in the west," the structure would look like the following:

^**imagery.command**.inspect
  ^*command-id* <unique-symbol>
  ^visual-spatial
    ^relative-visual-object <enemy-1-visual-object>
    ^base-visual-object <key-terrain-west-visual-object>
    ^direction.vector
    ^distance.scalar

The result structure includes the values. Likewise, if the agent desires the answer in qualitative terms, it includes the qualitative attribute without a value, and the resulting structure will have the closest qualitative value.

The imagery system uses Soar's subgoaling mechanism to implement spatial and visual imagery processing. When an agent augments the top-state ***imagery*** attribute with a command, the system proposes an imagery operator. If the operator is selected (it may not be selected because of current, more immediate task demands), then an operator no-change impasse occurs and Soar creates an imagery state structure. Imagery processing commences in this state and remains active until either the corresponding result structure of the command is created or another operator is selected in a superstate (e.g. in the Scout domain an operator to attend to the teammate's report).

The imagery subgoal includes operators to *compose, add, transform-om, transform-vb, generate, inspect*, and *attend-to-input-link.* With the exception of the *transform-vb*, each imagery command requires two decision cycles. The first decision cycle involves selecting the operator associated with the agent's imagery command and sending it to SVI by augmenting Soar's output-link. The second decision cycle attends to the results returned by SVI on Soar's input-link and creates ^**vs-wmem.imagery.result.***retrieved* structure in the top-state. For the compose and add commands, this second decision cycle (*attend-to-input-link)* creates the ^**vs-wmem.visual-object-instances.***visual-object* structures. The transform-vb operator may require additional decision cycles depending on the number of required attention-window shifts. In this case, the attend-to-input-link operator creates the appropriate output-link structures to affect the shift.

The primary purpose of each operator is to translate the agent's command into its primitive elements, augment missing structures with default values, and communicate with SVI via Soar's output- and input-link (Figure C-12a-b). For example, the structures illustrated in Figure C-11 are similar to the substructures augmented on the output-link except rather than specifying the relative- or base-visual-object identifier symbols, the primitive visual-ids and instance-ids are used. If information is missing, the selected operator augments the outgoing structure with its default values (e.g. topology defaults to disconnected, distance to 1.0, orientation to zero degrees, etc.).

After imagery finishes processing in SVI, the VisualLTMListener and ObjectMapListener augment the incoming **^what-link** and **^where-link** respectively. Both structures have a ^*recognize* and ^*result* attributes. The listeners automatically

update the two *recognize* attributes during each input phase. The **^what-link.***recognize* attribute has one ^*visual-object* structure for each salient object (either perceived or imagined) in the current scene. The **^where-link.***recognize* attribute has on ^*spatial* structure for each visual-object in the scene. The base-instance-id is the visual-object instance-id for the agent and the relative-instance-id is the salient object. Direction, distance, orientation, and relative size between the agent and perceived object are always provided on the incoming ^*recognize.spatial* attribute. So, for example, when the agent adds an imagined visual-object to the scene, the VLTMListener will automatically add its ^*what-link.recognize* visual-object structure. The ObjectMapListener will add its ^*where-link.recognize* spatial structure with the direction, distance, orientation, and size information specified relative to the agent. After each imagery operation, the listeners, at a minimum, augment each link's ^*result* attribute with the *command-id*. Other attributes include the vb-layer, shift, visual-feature, or spatial result similar to the top-state **^vs-wmem.imagery.result.***retrieved* structure previously discussed.

<table>
<tr>
<td>

^**output-link**<br>
^*imagery.command*<br>
^*compose*<br>
^*add*<br>
^*transform-om*<br>
^*transform-vb*<br>
^*inspect*<br>
# Note similar structure as in Figure C-11<br>
but using primitive values

</td>
<td>

^**input-link**<br>
^*what-link*<br>
^*recognize* # bottom-up,automatic<br>
^*result* # imagery results<br>
^*store/remove* # visual-ltm<br>
# store/remove<br>
^*where-link*<br>
^*recognize* # bottom-up,automatic<br>
^*result* # imagery results

</td>
</tr>
<tr>
<td>(a)</td>
<td>(b)</td>
</tr>
<tr>
<td>

^**input-link**.*what-link*<br>
^*recognize*<br>
^*visual-object*<br>
^*visual-id*<br>
^*instance-id*<br>
^*result*<br>
^*command-id*<br>
^*vb-layer* *# after generate command*<br>
^*shift true false # during vb transforms*<br>
^*visual-feature* # inspect command<br>
^*emergent-id*

</td>
<td>

^**input-link**.*where-link*<br>
^*recognize*<br>
^*spatial*<br>
^*relative-instance-id*<br>
^*base-instance-id*<br>
^*direction*<br>
^*distance*<br>
^*orientation*<br>
^*size*<br>
^*result*<br>
^*command-id*<br>
^*spatial* # inspect command

</td>
</tr>
<tr>
<td>(c)</td>
<td>(d)</td>
</tr>
</table>

**Figure C-12** Imagery Output-link and Input-link Working Memory Structure

# Bibliography

Anderson, J. R. (1978). Arguments concerning representations for mental imagery. *Psychological Review, 85*(4).

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review, 111*(4), 1036-1060.

Army, U. S. (2002). Field manual 3-20.98, reconnaissance platoon. Washington D.C.: Headquarters, Department of the Army.

Army, U. S. (2003). FM 6-0, mission command: command and control of Army forces. Washington, D.C.: Headquarters, Department of the Army.

Barkowsky, T. (2002). *Mental representation and processing of geographic knowledge - A computational approach*. Berlin: Springer-Verlag.

Barkowsky, T. (in press). Modeling mental spatial knowledge processing: An AI perspective. In F. Mast & L. Jäncke (Eds.), *Spatial processing in navigation, imagery, and perception*. Berlin: Springer.

Barsalou, L. W. (1999). Perceptual symbol systems. *Behavioral and Brain Sciences, 22*, 577-660.

Baylor, G. W. (1971). *A treatise on the mind's eye: an empirical investigation of visual mental imagery*. (Doctoral dissertation, Carnegie-Mellon, 1971).

Beazley, D., Ballabio, L., Fulton, W., Johnson, L., Köppe, M., Lenz, J., et al. (2002). Simplified wrapper and interface generator (SWIG). Retrieved October 15, 2005, from http://www.swig.org/

Best, B. J., Lebiere, C., & Scarpinatto, C. K. (2002). A model of synthetic opponents in MOUT training simulations using the ACT-R cognitive architecture. In *Proceedings of the Eleventh Conference on Computer Generated Forces and Behavior Representation*). Orlando, FL.

Biederman, I. (1987). Recognition-by-components: A theory of human image understanding. *Psychological Review, 94*(2), 115-147.

Buss, A. H. (2002). Component based simulation modeling with SimKit. In *Proceedings of the 2002 Winter Simulation Conference*, E. Yücesan, C.-H. Chen, J. L. Snowdon & J. M. Charnes (Eds.)). Miami, FL.

Buss, A. H., & Sánchez, P. J. (2005). Simple movement and detection in discrete event simulation. In *Proceedings of the 2005 Winter Simulation Conference*, M. E. Kuhl, F. B. Steiger, N. M. Armstrong & J. A. Joines (Eds.)). Miami, FL.

Byrne, P., Becker, S., & Burgess, N. (2007). Remembering the past and imagining the future: A neural model of spatial memory and imagery. *Psychological Review 114*(2), 340-375.

Chandrasekaran, B. (2006). Multimodal cognitive architecture: Making perception more central to intelligent behavior. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, (pp. 1508-1512). Boston, MA.

Chong, R. S., & Laird, J. E. (1997). Identifying dual-task executive process knowledge using EPIC-Soar. In *Proceedings of the Nineteenth Annual Conference of the Cognitive Science Society*, M. Shafto & P. Langley (Eds.)). Mahwah, New Jersey: Lawrence Erlbaum Associates.

Chown, E., Kaplan, S., & Kortenkamp, D. (1995). Prototypes, location, and associative networks (PLAN): Towards a unified theory of cognitive mapping. *Cognitive Science, 19*(1), 1-51.

Cohn, A. G., Bennett, B., Gooday, J., & Gotts, N. M. (1997). Qualitative spatial representation and reasoning with the Region Connection Calculus. *GeoInformatica, 1*(3), 275-316.

Eberly, D. H. (2005). *3D game engine architecture, first edition: Engineering real-time applications with Wild Magic*. San Francisco, CA: Morgan Kaufman Publishers Elsevier Inc.

Edwards, G., & Moulin, B. (1998). Toward the simulation of spatial mental images using the Voronoi¨ model. In P. Olivier & G. Klaus-Peter (Eds.), *Representation and Processing of Spatial Expressions* (pp. 163-184). Mahwah, New Jersey: Lawrence Erlbaum Associates, Inc.

Farah, M. J., Soso, M. J., & Dasheiff, R. M. (1992). Visual angle of the mind's eye before and after unilateral occipital lobectomy. *Journal of experimental psychology. Human perception and performance, 18*(1), 241-246.

Finke, R. A. (1989). *Principles of mental imagery*. Cambridge, MA: MIT Press.

Forbus, K. D., Neilsen, P., & Faltings, B. (1991). Qualitative spatial reasoning: the clock project. *Artificial Intelligence, 51*(1-3), 417–471.

Forbus, K. D., Tomai, E., & Usher, J. (2003). Qualitative spatial reasoning for visual grouping in sketches. *Paper presented at the 17th International Workshop on Qualitative Reasoning*. Brasilia, Brazil.

Funt, B. V. (1976). *WHISPER: A computer implementation using analogues in reasoning.* (Doctoral dissertation, The University of British Columbia, 1976).

Furnas, G. (1990). Formal models for imaginal deduction. In *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society*, (pp. 622-669). Hillsdale, NJ: Lawrence Erlbaum.

Furnas, G. (1991). New graphical reasoning models for understanding graphical interfaces. In *Proceedings of the CHI '91 Conference on Human Factors in Computer Systems*, (pp. 71-78).

Furnas, G., & Qu, Y. (2002). Shape manipulation using pixel rewrites, (at Visual Computation 2002). In *Proceedings of the Distributed Multimedia Systems 2002*, (pp. 630-639). San Francisco, CA.

Furnas, G., & Qu, Y. (2003). Using pixel rewrites for shape-rich interaction. In *Proceedings of the Human Factors in Computing Systems CHI2003 Conference*, (pp. 369-376). New York: ACM.

Furnas, G., Qu, Y., Shrivastava, S., & Peters, G. (2000). The use of intermediate graphical constructions in problem solving with dynamic, pixel-level diagrams. In *First International Conference on the Theory and Application of Diagrams: Diagrams 2000* (Vol. 1889, pp. 314-329). Edinburgh, Scotland, U.K.

Gelernter, H. (1959). Realization of a geometry theorem-proving machine. In *Proceedings of the International Conference on Information Processing*, (pp. 273-282). Unesco, Paris.

Gilden, D. L., Blake, R., & Hurst, G. (1995). Neural adaptation of imaginary visual motion. *Cognitive Psychology, 28*, 1-16.

Glasgow, J., & Papadias, D. (1992). Compuational imagery. *Cognitive Science, 16*, 355-394.

Grandin, T. (2006). *Thinking in pictures* (Second ed.). New York: Vintage Books.

Grush, R. (2004). The emulation theory of representation: Motor control, imagery, and perception. *Behavioral and Brain Sciences, 27*, 377-442.

Gunzelmann, G., & Lyon, D. R. (2007). Cognitive architectures: Valid control mechanisms for spatial information processing. In H. Schultheis, T. Barkowsky, B. Kuipers & B. Hommel (Eds.), *Technical Report #SS-07-01: AAAI Spring Symposium Series: Control Mechanisms for Spatial Knowledge Processing in Cognitive/Intelligent Systems* (pp. 23-28). Menlo Park, CA: AAAI Press.

Hebb, D. O. (1968). Concerning imagery. *Psychological Review, 75*(6), 466-477.

Helstrup, T. (1988). Imagery as a cognitive strategy. In M. Denis, J. Engelkamp & J. T. E. Richardson (Eds.), *Cognitive and Neuropsychological Approaches to Mental Imagery* (pp. 241-250). Dordecht / Boston / Lanchester: Martinus Nijhorff.

Hill, R. W. (1999). Modeling perceptual attention in virtual humans. In *Proceedings of the 8th Conference on Computer Generated Forces and Behavioral Representation*). Orlando, FL.

Hill, R. W., Han, C., & Van Lent, M. (2002). Perceptually driven cognitive mapping in a virtual urban environment. *AI Magazine, 23*(4), 67-69.

Itti, L. (2000). *Models of bottom-up and top-down visual attention.* (Doctoral dissertation, California Institute of Technology, 2000).

Jaczkowski, J. J. (2002, June 2002). Robotic technology integration for army ground vehicles. *Aerospace and Electronic Systems Magazine, 17,* 20-25.

Jones, R. M., Laird, J. E., Nielsen, P. E., Coulter, K. J., Kenny, P. G., & Koss, F. V. (1999). Automated intelligent pilots for combat flight simulation. *AI Magazine, 20*(1), 27-42.

Jonides, J., Lewis, R. L., Nee, D. E., Lustig, C. A., Berman, M. G., & Moore, K. S. (2008). The mind and brain of short-term memory. *Annual Review of Psychology, In Press*.

Kaplan, S., & Kaplan, R. (1982). *Cognition and environment*. Ann Arbor, MI: Ulrich.

Kettani, D., & Moulin, B. (1999). A spatial model based on the notions of spatial conceptual map and of object's influence areas. In C. Freksa & D. M. Mark (Eds.), *Spatial Information Theory. Cognitive and Computational Foundations of Geographic Information Science* (Vol. 1661/1999). Berlin: Springer-Verlag.

Kieras, D. E., & Meyer, D. E. (1997). An overview of the EPIC architecture for cognition and performance with application to human-computer interaction. *Human-Computer Interaction, 12*, 391-483.

Kosslyn, S. M. (1980). *Image and mind*. Cambridge: Harvard University Press.

Kosslyn, S. M. (1994). *Image and brain - the resolution of the imagery debate.* Cambridge: MIT Press.

Kosslyn, S. M., Alpert, N. M., Thompson, W. L., Maljkovic, V., Weise, S. B., Chabris, C. F., Hamilton, S. E., Rauch, S. L., & Buonanno, F. S. (1993). Visual mental imagery activates topographically organized visual cortex: PET investigations. *Journal of Cognitive Neuroscience, 5*, 263 - 287.

Kosslyn, S. M., & Pomerantz, J. R. (1977). Imagery, propositions, and the form of internal representations. *Cognitive Psychology, 9*, 52-76.

Kosslyn, S. M., Reiser, B. J., Farah, M. J., & Fliegel, S. L. (1983). Generating visual images: units and relations. *Journal of Experimental Psychology: General, 112*(2), 278-303.

Kosslyn, S. M., Thompson, W. L., & Ganis, G. (2006). *The case for mental imagery*. New York, New York: Oxford University Press.

Kuipers, B. (2000). The spatial semantic hierarchy. *Artificial Intelligence, 119*, 191-233.

Kurup, U. (2008). *Design and use of a bimodal cognitive architecture for diagrammatic reasoning and cognitive modeling* (Doctoral dissertation, Ohio State University, 2008).

Kurup, U., & Chandrasekaran, B. (2007). Modeling memories of large-scale space using a bimodal cognitive architecture. In *Proceedings of the Eighth International Conference on Cognitive Modeling*, R. L. Lewis, T. A. Polk & J. E. Laird (Eds.), (pp. 267-272). Oxford, UK: Taylor & Francis/Psychology Press.

Laird, J. E. (2008). Extending the Soar cognitive architecture. In *Proceedings of the Artificial General Intelligence Conference*). Memphis, TN.

Larkin, J. H., & Simon, H. A. (1987). Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science, 11*, 65-99.

Lehman, J., Laird, J., & Rosenbloom, P. (2006). A gentle introduction to Soar, an architecture for human cognition: 2006 Update. Retrieved Jan 2, 2006, from http://ai.eecs.umich.edu/soar/sitemaker/docs/misc/GentleIntroduction-2006.pdf

Lightweight Java game library (LWJGL). (2008). Retrieved October 15, 2005, from http://lwjgl.org/

Marr, D. (1982). *Vision*. San Francisco: Freeman.

Mat Jafri , M. Z., & Deravi , F. (1994, 2 November 1994). Efficient algorithm for the detection of parabolic curves. In *Proceedings of the Proceedings of SPIE - Vision Geometry III*, R. A. Melter & A. Y. Wu (Eds.), (pp. 53-61). Boston, MA, USA SPIE--The International Society for Optical Engineering.

Mellet, E., Bricogne, S., Tzourio-Mazoyer, N., Ghaem, O., Petit, L., Zago, L., et al. (2000). Neural correlates of topographic mental exploration: The impact of route versus survey perspective learning. *NeuroImage, 12*, 588-600.

Moran, T. P. (1973). *The symbolic imagery hypothesis: An empirical investigation via a production system simulation of human behavior in a visualization task.* (Doctoral dissertation, Carnegie-Mellon, 1973).

Mukerjee, A. (1998). Neat versus scruffy: A review of computational models for spatial expressions. In P. Olivier & G. Klaus-Peter (Eds.), *Representation and Processing of Spatial Expressions* (pp. 1 - 31). Mahwah, New Jersey: Lawrence Erlbaum Associates, Inc.

Newell, A. (1990). *Unified theories of cognition*. Cambridge, Massachusetts: Harvard University Press.

Newell, A., & Simon, H. A. (1972). *Human problem solving*. Englewood Cliffs, N.J.: Prentice-Hall.

Newell, A., Yost, G. R., Laird, J. E., Rosenbloom, P. S., & Altmann, E. (1991). Formulating the problem space computational model. In R. F. Rashid (Ed.), *Carnegie Mellon Computer Science: A 25-Year commemorative*. Reading, MA: ACM-Press (Addison-Wesley).

Norman, J. (2000). Differentiating diagrams: A new approach. In M. Anderson, P. Cheng & V. Haarslev (Eds.), *Theory and Application of Diagrams* (Vol. 1889, pp. 105-116). Berlin Heidelberg Springer-Verlag.

Olson, C. F. (1999). Constrained Hough transforms for curve detection. *Computer Vision and Image Understanding, 73*(3), 329-345.

Palmer, S. E. (1999). *Vision science photons to phenomenology*. Cambridge, Massachusetts: The MIT Press.

Peronnet, F., Farah, M., & Gonon, M. (1988). Evidence for shared structures between imagery and perception. In M. Denis, J. Engelkamp & J. T. E. Richardson (Eds.), *Cognitive and Neuropsychological Approaches to Mental Imagery* (pp. 357-362). Dordecht / Boston / Lanchester: Martinus Nijhorff.

Pinker, S. (1988). A computational theory of the mental imagery medium. In M. Denis, J. Engelkamp & J. T. E. Richardson (Eds.), *Cognitive and Neuropsychological Approaches to Mental Imagery* (pp. 17-32). Dordecht / Boston / Lanchester: Martinus Nijhorff.

Podgorny, P., & Shepard, R. N. (1978). Functional representations common to visual perception and imagination. *Journal of Experimental Psychology: Human Perception and Performance, 4*, 21-35.

Pylyshyn, Z. (1973). What the mind's eye tells the mind's brain: A critique of mental imagery. *Psychological Bulletin, 80,* 1-24.

Pylyshyn, Z. (1981). The imagery debate: Analogue media versus tacit knowledge. *Psychological Review, 88*, 16-45.

Pylyshyn, Z. (2001). Visual indexes, preconceptual objects, and situated vision. *Cognition, 80*, 127-158.

Pylyshyn, Z. (2002). Mental Imagery: In search of a theory. *Behavioral and Brain Sciences, 25*, 157-238.

Ritter, G. X., & Wilson, J. N. (1996). *Handbook of computer vision algorithms in image algebra*. Boca Raton, Florida: CRC Press, Inc.

Schultheis, H., Bertel, S., Barkowsky, T., & Seifert, I. (2007). The spatial and the visual in mental spatial reasoning: An ill-posed distinction. In T. Barkowsky, M. Knauff, G. Ligozat & D. R. Montello (Eds.), *Spatial Cognition V - Reasoning, Action, Interaction* (pp. 191–209). Berlin: Springer Verlag.

Shepard, R. N., & Metzler, J. (1971). Mental rotation of three-dimensional objects. *Science, 171*, 701-703.

Shreiner, D., Woo, M., Neider, J., & Davis, T. (2006). *OpenGL programming guide*. Upper Saddle River: Addison-Wesley.

Simon, H. A. (1996). The patterned matter that is mind. In D. M. Steier & T. M. Mitchell (Eds.), *Mind Matters A Tribute to Allen Newell* (pp. 407-431). Mahwah, New Jersey: Lawrence Erlbaum Associates, Inc.

Soar. (2008). Retrieved October 15, 2005, from http://sitemaker.umich.edu/soar/home

St. Amant, R., Riedel, R., Ritter, F., E., & Reifers, A. (2005). Image processing in cognitive models with SegMan. In *Proceedings of the HCI International 2005*). Las Vegas, NV: Lawrence Erlbaum Associates.

The standard widget toolkit (SWT). (2007). Retrieved October 15, 2005, from http://www.eclipse.org/swt/

Stevens, A., & Coupe, P. (1978). Distortions in judged spatial relations. *Cognitive Psychology, 18*, 422-437.

Sun, R., Slusarz, P., & Terry, C. (2005). The interaction of the explicit and the implicit in skill learning: A dual-process approach. *Psychological Review, 112*(1), 159-192.

Tabachneck-Schijf, H. J. M., Leonardo, A. M., & Simon, H. A. (1997). CaMeRa: A computational model of multiple representations. *Cognitive Science, 21*(3), 305-350.

Tambe, M., Johnson, W. L., Jones, R. M., Koss, F. M., Laird, J. E., Rosenbloom, P. S., & Schwamb, K. B. (1995). Intelligent agents for interactive simulation environments. *AI Magazine, 16*(1), 15-39.

Thompson, W. L., Kosslyn, S. M., Hoffman, M. S., & van der Kooij, K. (in press). Inspecting visual mental images: Can people "see" implicit properties as easily in imagery and perception? *Memory & Cognition*.

Tolman, E. C. (1948). Cognitive maps in rats and men. *Psychological Review, 55*(4), 189-208.

Tootell, R. B. H., Silverman, M. S., Switkes, E., & De Valois, R. L. (1982). Deoxyglucose analysis of retinotopic organization in primate striate cortex. *Science, 218*, 902-904.

Tschumperlé , D. (2008). The CImg library: C++ template image processing toolkit. Retrieved January 5, 2008, from http://cimg.sourceforge.net/

Tye, M. (1991). *The imagery debate*. Cambridge, MA: MIT Press.

Ullman, S. (1996). *High-level vision object recognition and visual cognition*. Cambridge, Massachusetts: The MIT Press.

Ungerleider, L. G., & Mishkin, M. (1982). Two cortical visual systems. In D. J. Ingle, G. M.A. & R. J. W. Mansfield (Eds.), *Analysis of visual behavior* (pp. 549-586). Cambridge, MA: MIT Press.

Watson, J. B. (1913). Psychology as the behaviorist views it. *Psychological Review, 20*, 158-177.

Weaver, M. (1993). *An active symbol connectionist model of concept representation.* (Doctoral dissertation, University of Michigan, 1993).

Wintermute, S., & Laird , J. E. (2008). Bimodal spatial reasoning with continuous motion. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-08)*). Chicago, Illinois.

Wray, R. E., Laird, J. E., Nuxoll, A., Stokes, D., & Kerfoot, A. (2005). Synthetic adversaries for urban combat training. *AI Magazine, 26*(3), 82-92.

Yamamoto, K. (1996). Visulan: A visual programming language for self-changing bitmap. In *Proceedings of the International Conference on Visual Information Systems*, (pp. 88-96). Melbourne, Australia.

Yeap, W. K., & Jefferies, M. E. (1999). Computing a representation of the local environment. *Artificial Intelligence, 107*, 265-301.