The Director's Apprentice: Animating Figures in a Constrained Environment

by

Garfield John Ridsdale

B.Sc., University of British Columbia, 1974

M.Sc., Queen's University, 1980

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

in the School

of

Computing Science

SIMON FRASER UNIVERSITY

June 1987

Name: Garfield John Ridsdale
Degree: Ph.D.
Title of Thesis: The Director's Apprentice: Animating Figures in a Constrained Environment
Examining Committee:
Chairperson:

Dr. Thomas W. Calvert
School of Computing Science
Senior Supervisor

Dr. Veronica Dahl
School of Computing Science
Internal External Examiner

Dr. Brian V. Funt
School of Computing Science

Dr. Norman I. Badler
Dept. of Computer Science
University of Pennsylvania
External Examiner

Dr. Nick J. Cercone
School of Computing Science

Dr. John Dill
Adjunct Professor
School of Computing Science

2 July 87
Date Approved

PARTIAL COPYRIGHT LICENSE

I hereby grant to Simon Fraser University the right to lend my thesis, project or extended essay (the title of which is shown below) to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users. I further agree that permission for multiple copying of this work for scholarly purposes may be granted by me or the Dean of Graduate Studies. It is understood that copying or publication of this work for financial gain shall not be allowed without my written permission.

Title of Thesis/Project/Extended Essay

The Director's Apprentice:
Animating Figures in a Constrained Environment

Author:

(signature)

G. J. Ridsdale

(name)

2 July 87

(date)

Abstract

This thesis describes the "Director's Apprentice", a project for producing computer graphic films of animated human characters. The application of scene-level constraints on a figure animation system is discussed, along with extensions to logic programming techniques for quantitative reasoning. These techniques are applied to the problem of defining a hierarchy of planning constraints that guide the actions of the characters in the depicted scene. A prototype implementation of these concepts is described that is capable of generating character movements including walking, running and climbing stairs in a cluttered environment.

Acknowledgements

I would like to express my gratitude to the many people whose help, support and encouragement have made this thesis possible. First and foremost, thanks go to Dr. Thomas W. Calvert for his continued support of my research, and of my personal development as a student and scientist. Without his support, advice, constructive criticism and fresh viewpoints, I would have had great difficulty completing my work.

Drs. John Dill, Brian Funt and Nick Cercone all provided help and advice on the technical matters of the thesis. Mr. Ed Bryant also deserves thanks for his help with many day-to-day problems encountered while developing the prototype animation system.

This research has been largely funded through grants obtained from the Natural Sciences and Engineering Research Council of Canada, and from the Science Council of British Columbia.

Special thanks go to my dear wife, Valerie, for her boundless patience and support during the past five years, and to my parents Reg and Peg for their belief in me.

Table of Contents

List of Figures

Chapter 1:
Introduction

Problem Specification

This thesis describes the "Director's Apprentice", a project for animating human characters in a constrained environment.

Applications of figure animation techniques may be found in entertainment [Calvert 80], education [Calvert 82] and the simulation of complex environments [Badler 85]. However, existing figure animation techniques are so cumbersome and laborious that only short high-budget productions, such as a television commercials or music videos, can justify the labour cost [e.g., Kroyer 86].

Most existing methods of computerized figure animation use some form of keyframing (e.g., [Sturman 84], [Steketee 85]). In keyframing, the animator begins by specifying a sequence of positions for each of the characters. Once specified, the

computer fills in the intervening frames using a numerical technique, typically *parametric keyframe interpolation.* Setting up the keyframes is a highly-skilled and time-consuming operation. For this reason, and despite many potential applications in science and the arts, computerized figure animation is not widely used.

Parametric interpolation works well if the keyframes are spaced closely together in time. But, this process would be improved if fewer keyframes were needed, since this would mean less work for the animator. Fewer keyframes would speed up revisions, make longer works feasible, and make figure animation techniques available to less-skilled artists: a fifty percent reduction in keyframes could save months of work. What is needed is a system which will only generate reasonable actions, that is, actions that are sensitive to the spatial context of the characters and to the semantic context of the scene. If a faster and easier method can be developed for specifying the character actions in a scene, figure animation may potentially become a common medium of communication, just as still pictures and sketches are today.

This requires *scene knowledge.* Scene knowledge is the general commonsense knowledge of scenes possessed by any competent animator, along with the details found in a script. Commonsense knowledge includes those things which characters tend to do (walk quickly when nervous, walk slowly when tired, approach people that they like), as well as those which characters tend *not* to do (walk through solid objects or

approach that which they fear).

One of the richest sources of scene knowledge is the experience of theater directors. These experts in scene action know a repertoire of rules which, if obeyed, can make the difference between a well-performed scene and one that looks "amateurish". Most people are familiar with what happens when these rules are ignored: minor characters stand in front of speaking characters, conversation bounces from one side of the stage to the other, characters face away from the audience while speaking, and so on. These principles are well known, and described in theater textbooks (e.g. [Brown 36], [Allensworth 82] and [Benedetti 85]).

For example, areas of the stage have different "strengths"; strong areas attract attention. Generally, downstage areas are stronger than upstage areas, and center areas stronger than either the right or left.

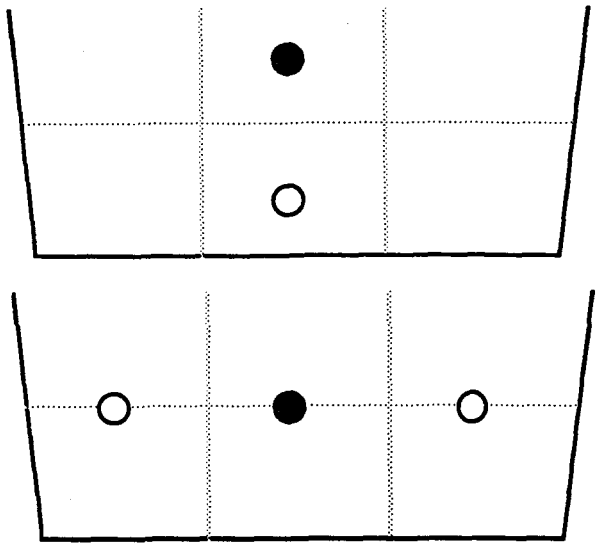

Figure 1-1 Position Strengths
Black circles are in
stronger areas than white circles

It follows that a program animating a scene where one character is more important than the others should have the main character gravitate to a downstage-central position.

Some *movements* are stronger than others, also. Brown([Brown 36]) writes:

> Strong movements are forward movements, that is toward the audience. ...
>
> The forward movements, listed with the strongest first, are up center to down center, up right or up left to down center, the full stage diagonals, ... and the movements from right or left to center parallel to the curtain line at any level.

These movements are illustrated in Figure 1-2.

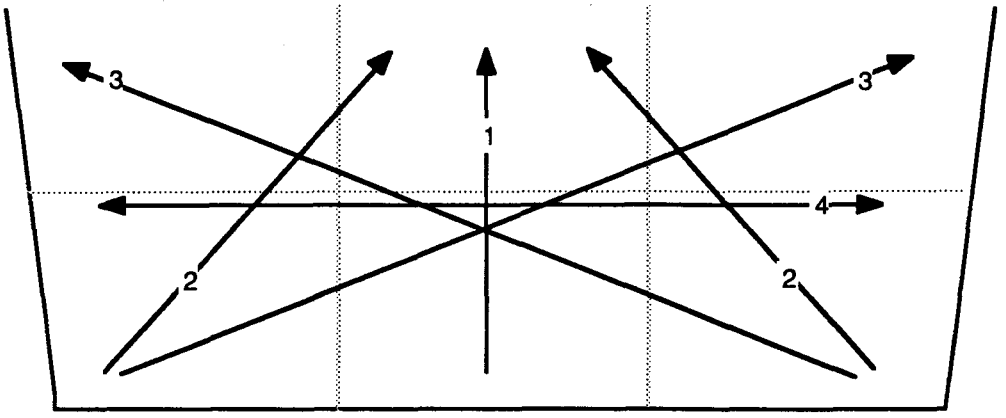

Figure 1-2 Strong movements (In descending order)

If a script specifies only vague actions, such as "Miss Kitty exits", when there are many different exit points, it will be up to the animation program to choose a strong or weak movement that is appropriate to the character. For example, the lead character in a scene will typically use strong movements to attract attention to himself while secondary characters will use weaker movements.

Another important concept in directing is *focus*, using stage positions to control attention. Focus concerns the use of subtle character action to shift audience attention from one character to the next, generally one step ahead of the next change of speaker ([Allensworth 82]). This is done so that the audience will have time to settle its "focus" on a character *before* he begins to speak; otherwise, his first few words or gestures may be lost to those of the audience who watching someone else.

In *focus by position*, a new main character (who is just about to begin speaking) receives focus by moving to a stronger stage position, relative to the current main character (who are about to finish speaking).

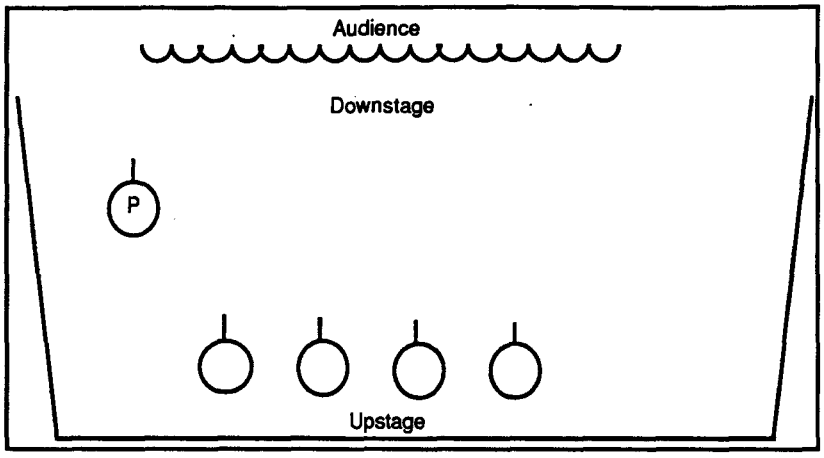

Figure 1-3 Focus by position

In *actual line focus*, the new main character receives focus from the supporting characters by having them line up in a virtual "arrow" pointing at him. In Figure 1-4, note that some of the other characters are in "stronger" locations than the main character, yet he receives the focus because he is set off from them.

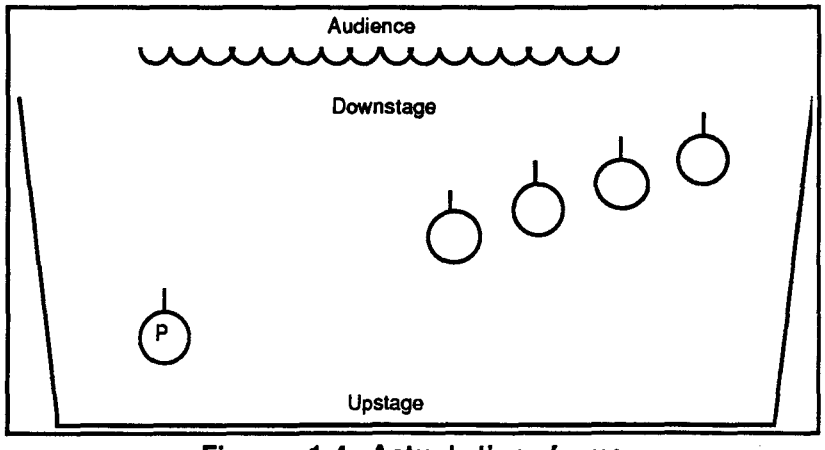

Figure 1-4 Actual line focus (after [Allensworth 82])

In *visual line focus*, the main character receives focus by having the other characters turn to face him.

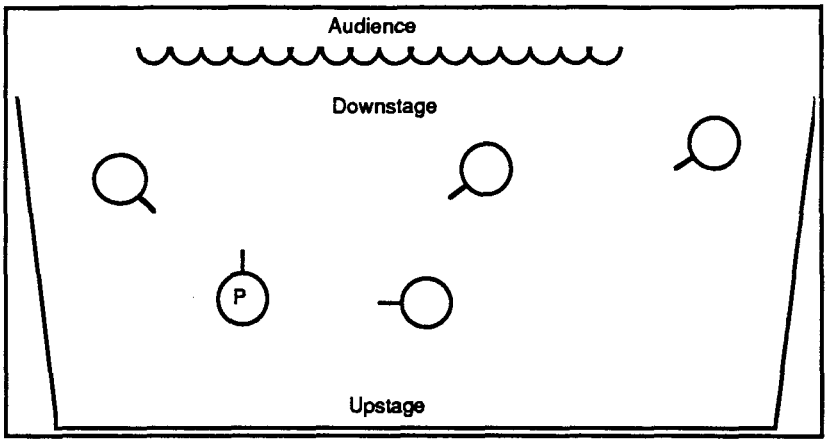

Figure 1-5 Visual Line Focus

Script interpretation is the principal job of a director (whether of a play, film or cartoon), i.e., filling in enough of the missing details to complete the action. Script actions include gross body movements such as "Phillip exits stage left", and detailed actions such as "Mary turns to face Jane". Clues to a character's locomotion or *gait pattern* may also be found, as it is affected by personality, weight, strength, mood, age, anatomy and some diseases. For example, a script reference such as "Ron, a sick, tired old man, moves downstage" contains several clues that can be used by an animation program to select an appropriate gait, although this goes beyond conventional keyframe animation techniques.

Since one of the goals of this thesis is to develop a framework for describing the rules of good direction — one that can grow and improve over time — the prototype implementation is titled the "Director's Apprentice": an "expert system" for scene direction (see [Ridsdale 86]). The approach taken in developing the Director's Apprentice is to apply logic-based constraint planning to derive the parameters needed to animate walking figures in a scene. The scene may have several characters, and a number of stage props that are treated as obstacles to be navigated. The key problem is to find a way to easily define and revise the scene constraints.

Goals

The principal goals of the research behind this thesis are as follows:

1. to investigate the significance of scene-level constraints in implementing a useful figure animation system;
2. to investigate extensions to logic programming for dealing with quantitative reasoning;
3. to investigate the application of an extended logic programming tool for defining a hierarchy of constraints on animated figures;

4. to develop a prototype animation system, based on extended logic programming, that can be used to study the use of scene constraints in figure animation, and to assess whether it is qualitatively better than conventional (keyframe-based) figure animating systems.

Thesis Plan

Chapter 2 is a review of some projects in graphics, AI and robotics whose themes share common ground with the research of this thesis. The graphics projects include research in kinematic animation (e.g., [Calvert 82], [Badler 85], [Girard 86]), dynamic animation (e.g., [Wilhelms 86]) and planning hierarchies (e.g., [Zeltzer 82]). Relevant AI research includes that on logic programming (e.g., [Cohen 85]), conflicting knowledge (e.g., [Shortliffe 76]) and planning (e.g., [Fikes 71] and [Sacerdoti 77]). The aims and achievements of these projects are compared to those of the Director's Apprentice.

Chapter 3 is a discussion of conflicting knowledge in figure animation and its relation to constraint satisfaction and planning. Some relevant constraint research (e.g., [Stefik 81] and [Borning 79]) is described, and some issues in constraint formulation, propagation and satisfaction are discussed in the context of animating a scene. Some of the constraint satisfaction techniques applied in the Director's Apprentice project are described.

Chapter 4 is a discussion of the implementation of the prototype Director's Apprentice and includes a description of a formal language for defining the discrete and real-valued constraints on moving figures. The graphics tools used to display the action as walking three-dimensional figures are also described. These tools are based on the use of constrained inverse kinematics, which is fast and sufficiently versatile to handle a wide range of movements in real time.

Chapter 5 discusses the conclusions of this thesis, in relation to the goals set out in this chapter. The implications of the prototype project are discussed along with some suggestions for future research.

Finally, the appendix describes the grammar of the Scene Constraint Language.

Chapter 2:
Relationship to Other Work

In this chapter some research in computer graphics and in AI that relates to the aims of this thesis is reviewed . While none of the projects described have the same goals as the Director's Apprentice, many of them contain useful ideas for improving the power and intelligence of a scene animation system.

Approaches to Figure Animation

This first section reviews some previous attempts to animate figure in a flexible, adaptable way. There has been interest in the problem of animating human figures since at least the late 1970's ([Burntyk 76], [Badler 78], [Calvert 80], [Dooley 82], [Girard 86]). Since then the dominant approach to describing human animation has been *kinematics*. More recently, however, some research has been done in adding *constraints* ([Zeltzer 82]) and *dynamics* ([Wilhelms 86], [Armstrong 86]) to improve the power of the animation tools.

Kinematics

In kinematic animation([Badler 78], [Herbison-Evans 78], [Calvert 80], [Calvert 82], [Dooley 82], [Fetter 82], [Kochanek 82], [Calvert 83], [Sturman 84], [Steketee 85], [Kroyer 86]) the motion of the elements in the scene is planned out directly by the animator, and the computer animation system plays no part to ensure that the action is reasonable.

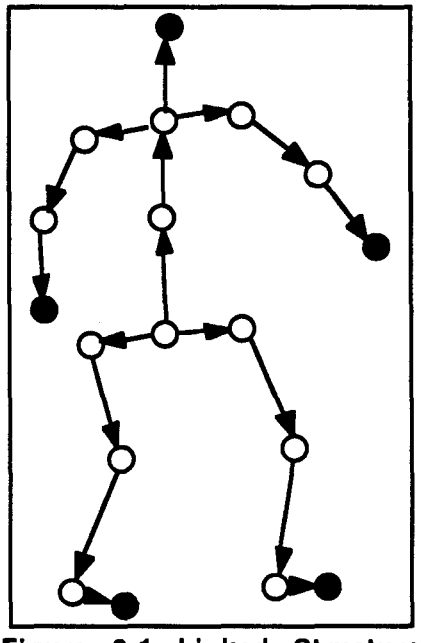

Figure 2-1 Linked Structure
A linked structure representing the parts of the human body

In human animation the body is usually represented as a linked structure with the terminal nodes (hands, feet, etc.) referred to as *end-effectors*, as illustrated in Figure 2-1. Kinematic figure animation is achieved by varying the joint-angles values, and the

location of the root of the structure in three-dimensional space. (The "root" in Figure 2-1 is the base of the spine.) In forward kinematics, the system works *forward* from the joint angles to determine the unique configuration of the end-effectors. For example, the root location plus the angles of the right leg will uniquely determine the location of the right foot, but the converse is not true; the location of the right foot does not uniquely determine the joint angles of the leg.

A paper by Calvert et al ([Calvert 82]) discusses the kinematic specification of human figure animation by the use of a dance notation called "Labanotation". Dance notation systems have the advantage of being well developed and internationally standardized; Labanotation, in particular, is well suited to specifying the sequence of body positions that a dancer executes in performing a piece of choreography. (See also [Singh 83], [Ryman 83]) and Badler and Smoliar ([Badler 79]). In contrast to the early approaches of Badler and Calvert, the Director's Apprentice project is less concerned with specifying complex movements in a simple environment (for example, a ballerina performing alone on a dance floor) than automatically generating *simpler* movements in a *complex* environment (for example, several people walking and gesturing on a stage set).

Another popular method of specifying animation kinematically is to take live data from, for example, a runner or dancer, and to use the data to drive a graphics display. This is

called *rotoscoping*, and may use two-dimensional data from a motion-picture film, or three-dimensional data from goniometers or video scanning equipment. See [Calvert 80] and [Ginsberg 82] for discussions of this technique. Perfectly life-like motion can be obtained this way, but at the cost of poor flexibility after the data is collected, in the sense that the animation has been specified at such a detailed level that adapting it for another sequence may be prohibitively difficult.

One problem with systems based on forward kinematics is that they tend to be laborious. This is because not even the simplest of environmental constraints (such as "people do not walk through other people" and "neither foot should go through the floor") are dealt with automatically. Nor can the actual paths of the hands, feet or head be planned in advance. As a result, repeated corrections to the details of the animation may be required.

In forward kinematics, the path of the end-effector through space is left unconstrained when the joint angles are interpolated between keyframes. In *inverse* kinematic animation([Girard 85], [Zeltzer 82]), the animator specifies the locations of the end-effectors in Cartesian coordinates, and the system determines the corresponding joint angles . Much of the work in this area comes from robotics research, e.g. [Lozano-Pérez 80], [Paul 81], [Lee 82], [Duffy 84], and [ElMaraghy 86].

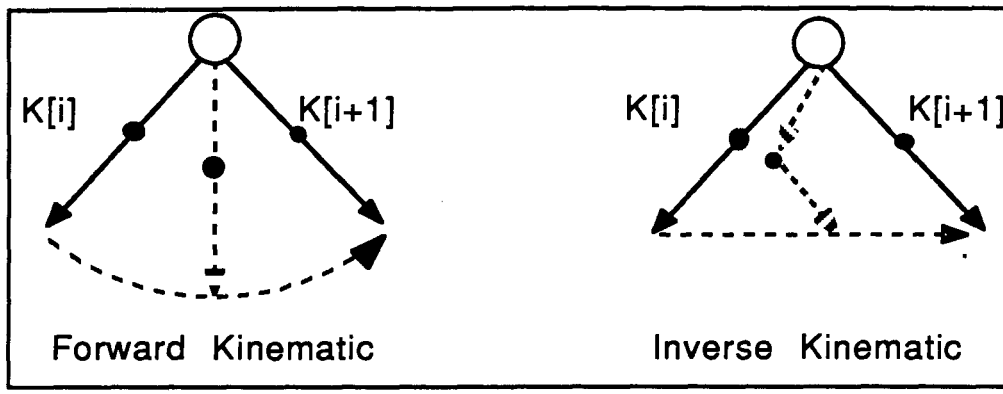

Figure 2-2 Kinematic Paths
(left) curving path swept by end-effector when
(right) straight-line path is required

Figure 2-2 depicts two keyframe positions of a robot or human arm. Since the only parameter that changes between keyframes K[i] and K[i+1] is the value of the shoulder joint angle, the end-effector will trace a curving path through space if the keyframes are interpolated using forward kinematics. However, the desired path of the end-effector between keyframes might be a straight line which cannot be expressed directly by forward kinematics. For a second example, if the action consists of a foot moving through a complex gait pattern (as in walking, running or skipping) then it would be more convenient to specify the motion of the end-effector directly than to have to manipulate joint angles. A third example is a hand making a complex gesture such as scratching or picking up a coffee cup .

If there were a general solution to the inverse kinematics problem, then as long as the desired path of the end-effector were known, the corresponding joint angles could be

calculated automatically. However, no such general solution exists (nor could exist). The problem is that the inverse kinematics problem is underconstrained, since there may be many sets of joint angles for any end-effector position. As with any underconstrained problem, inverse kinematic animation requires a procedure to select the *best* solution from among those that satisfy the end-effector constraints. ("Best" here means satisfying as many as possible of the figure's other constraints as well.)

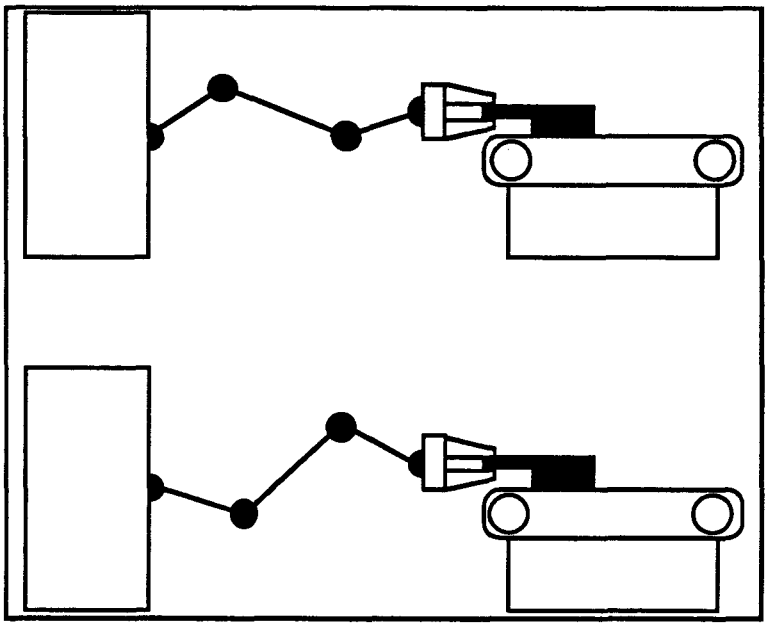

Figure 2-3 joint angle configurations for the same end-effector position

One approach, used by Girard and Maciejewski ([Girard 85]) is to generate an "inverse Jacobian" matrix of linear approximations to the (nonlinear) functions relating joint

position to joint angle. Girard applied this technique to generating complex gait patterns for four-legged (and many-legged) animals. When combined with a sophisticated interface for describing gait patterns (trotting, galloping, prancing, and others) this method works well for generating believable films of animal motion.

Solutions for general linkages, such as the inverse Jacobian method, are good for non-realtime animation of robots, but are too slow for real-time animation of human figures. This is because general-linkage methods do not use the natural constraints of human anatomy to make the constraint satisfaction easier; for example, there is no need to try to compute three degrees of freedom for the elbow since it is a single-degree-of-freedom hinge. Even joints that do have three degrees of freedom, such as the ankle, have anatomical limitations that can be exploited. There is certainly no need to deal with *four* degrees of freedom (including translation) for each joint in the body as Girard's technique does, since all adjacent pair of joints are a constant distance apart.

The *reach hierarchy* of Korein and Badler ([Korein 82]) is another inverse kinematic solution. (See Figure 2-4.) This method applies the rule "move each joint the smallest amount between keyframes that will allow the end-effector to reach the goal", assigning angle values to the joints outwardly from the root to the end-effector.

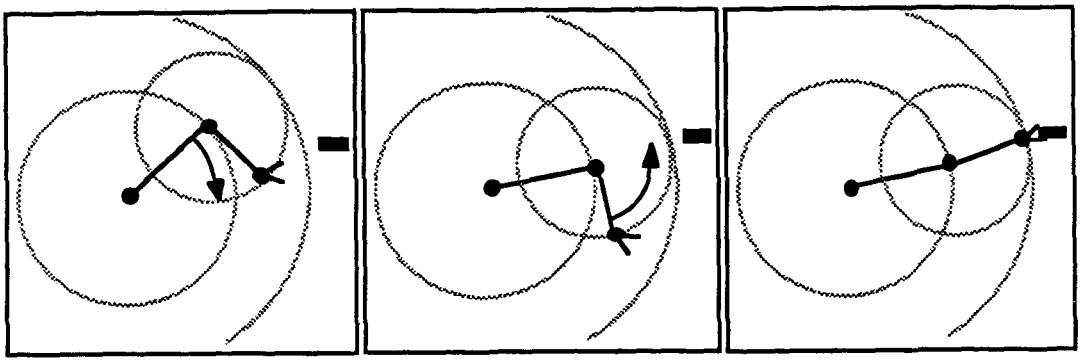

Figure 2-4 Reach Hierarchy Approach: Each step puts the reach area of the next limb segment within reach of the goal.

The reach hierarchy method is an inverse kinematic solution since the goal (the absolute position of the last joint) is known, and the joint *angles* are the result. Since the problem is constrained by joint limits and circles of reach, it is faster than calculating inverse Jacobians. However, it does not account for human constraints of coordinated movement or habit, and so may be difficult to apply to walking. A more recent paper by Badler et al ([Badler 85]) discusses a system for simulating movements of people in a complex environment, which includes a reach-hierarchy mechanism for animating arm movements (as in [Korein 82]) and a kinematic animator for the whole body (as in [Calvert 80]). But apart from the ability to reach or not reach certain objects, the character's environment places no constraints on the animation.

In summary, while kinematic animation is easy to implement and provides the most flexibility to the animator — in the sense that anything, including impossible actions can be specified — the labour cost is high. This is because any change to the scene — such

as a re-arrangement of the furniture — may require reworking all of the animation. This limits its feasibility to applications where both the time and the expense are justified.

Dynamics

A dynamic animation system provides the animator with automatic aids for constraining the figures to obey certain rules of Newtonian mechanics between keyframes. In the system described by Jane Wilhelms ([Wilhelms 86]) some aspects of human dynamics can be modelled and simulated, including the force of gravity on a falling figure, the inertia of the arms, and the effects of friction on a foot sliding along a floor. A faster method of calculating these effects of has been developed by researchers at the University of Alberta ([Armstrong 86]). The current thesis research of Armin Bruderlin at SFU concentrates on the use of a rule-based system (in this case a finite state machine) to control application of dynamic equations to locomotion. This work is promising but incomplete.

In all of these systems, the figure is a passive element of the environment. Only the PODA system of Girard and Maciejewski ([Girard 85]) deals directly with the dynamics of coordinated movements. Although in the long term, some attention to dynamic constraints would be a useful addition to the Director's Apprentice project, it is currently

more concerned with the kinematics of intentional, coordinated movements, such as walking from one place to another in a cluttered environment.

Constraint Hierarchies

In a 1982 paper ([Zeltzer 82]) David Zeltzer examines figure animation from several *abstraction levels*, and proposes that a goal-directed planning system be built along the lines suggested by Figure 2-5:

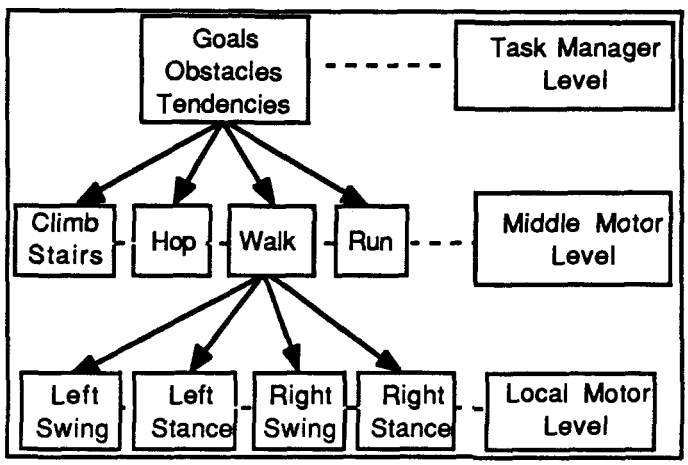

Figure 2-5 Zeltzer's Hierarchy

The three abstraction levels in Zeltzer's planning hierarchy are, first, the *task* level, which determines the sequence of skills to be performed; second, the *middle motor* level, which provides the definitions of the skills; and, third, the *local motor* level, which drives the figure by breaking down the skills into actual joint movements. Zeltzer's approach to figure animation is similar to the hierarchical planning approach of

[Sacerdoti 77]. Only the local motor level had been fully implemented when this paper ([Zeltzer 82]) was published.

The Director's Apprentice is built on a *script interpreter* and a *gait interpreter.* To use Zeltzer's terms, the script interpreter is similar to the task manager, while the gait interpreter combines the middle and local motor controllers. Both have been implemented in the prototype Director's Apprentice.

A paper by researchers at the University of Toronto ([Drewery 86]) discusses some issues in frame-based goal direction, based on English motion verbs. This work is at a preliminary level.

Summary of Figure Animation Systems

The forward kinematic approaches of Badler and Calvert both provide the animator with the ability to specify the arbitrarily complex sequences of figure position keyframes. What these approaches lack is the ability to constrain what happens between the keyframes to that which is likely or even humanly possible. (Both systems constrain joint angles, but this will not prevent one leg from passing through another or through the floor). The inverse kinematic approaches of Korein or Girard allow the animator to constrain the positions of end-effectors between keyframes, but these constraints do not

adjust automatically to changes in the character's environment. The dynamic approaches of Wilhelms or Armstrong address many important constraints on character motion but not global issues such as coordination or planning. Zeltzer's constraint hierarchy approach is the only one similar to that of the Director's Apprentice, but this was only a skeleton of a complete system when published in 1982.

All of these systems have deficiencies regarding the scene animation of human characters:

1. the inability to deal automatically with changes in the environment;
2. the inability to animate on the basis of the personal characteristics of the characters; and
3. the inability to resolve multiple, conflicting, time-varying constraints imposed on a character by its environment.

This thesis contends that high-level planning is the key to animating characters in a complex environment, and that a hierarchical refinement of constraints is the most efficient way to implement a plan. To my knowledge, this has not been done before.

Representing Knowledge with Logic Programming

This section discusses some research in the use of logic programming as a tool for representing knowledge.

The topic of representing knowledge in a computable form has occupied the AI community for decades, and remains controversial. Papers that survey and compare formal representations for knowledge include [Hayes 74],[Barr 80], [Delgrande 86] and [McCalla 83]. Proponents of various formalisms, including semantic networks, "frames" and logic programming, argue their relative merits in terms of representational adequacy, semantic clarity and other characteristics. Despite the controversy, there are three points of agreement that exist among knowledge representation researchers as to the *minimum* requirements of any efficient knowledge representation scheme: it must have a formal notation, be semantically clear and be efficiently computable. The central argument here is that logic programming meets all of these criteria and can be applied successfully to describing scene constraints; however no claim is made as that it is a good choice for natural language understanding, vision or story-telling.

A Formal Notation

A formal notation is one where an exact semantic model, or precise meaning, can be assigned to any well-formed statement in that notation. For example, the notation of algebraic expressions is formal since the meaning of any legal combination of variables and operators is clearly defined. Informal notations include English text and pencil sketches.

Predicate logic is a formal notation and can be used to represent the constraints on moving figures as described in Chapters 3 and 4. One advantage of logic programming is that the important characteristic of *inconsistency* is well-defined. A knowledge base is inconsistent when there could not be a model of reality that satisfies it. For example, the following statements are consistent:

1. $\forall x[(\text{is-onstage}(x) \wedge \neg\text{is-offstage}(x)) \vee (\text{is-offstage}(x) \wedge \neg\text{is-onstage}(x))]$[1]

2. is-onstage(John)

But with the addition of:

3. is-offstage(John)

---

1 Read "for all substitutions of variable x, character x is either onstage and not offstage, or he is offstage and not onstage".

the resulting set of statements is inconsistent. Statements 2 and 3 imply that John is both onstage and offstage, while statement 1 implies that this can never happen. Efficient techniques exist to detect inconsistencies in a logic programming-based knowledge representation ([Cohen 85]).

Semantic Clarity

Semantic clarity in a knowledge representation scheme implies not only that there is a well-defined meaning for each representation, but also that the meaning is clear and unambiguous to human readers. Semantic clarity is one of the most hotly-debated topics in knowledge representation, particularly with respect to the merits of predicate logic ([Hayes 74], [Hayes 77]). Nonetheless, predicate logic has been studied and developed since the time of Aristotle, and is widely known and understood outside of the computer science community. The advantage of this for the Director's Apprentice is that the knowledge base can be understood by those lacking a programming background, provided that they have a knowledge of logic.

For example, suppose that a scene is to be represented consisting of three individuals, "Fred", "Mike" and "Mary", and a single predicate "likes". Let "A" represent the statement "(Fred likes Mary)" and let "B" represent the statement "(Mary likes

Mike)". The meaning of the following statements is clear in terms of the world being represented:

Predicate Logic	English
$A \wedge B$	It is true that Fred likes Mary and that Mary likes Mike
$A \vee B$	It is either true that Fred likes Mary or that Mary likes Mike.
$\forall x,y,z$ ((x likes y)$\wedge$(y likes z)) $\supset$ (x likes z)	Transitivity does not hold over the predicate "likes".

The semantics of any combination of primitive logical operators ("and", "or", "not", "implies") and quantifiers ("for all", "there exists") can be analyzed and understood without the use of a computer. This quality of being widely and easily understood is an important advantage of logic programming as a knowledge representation method.

Efficient Computability

Efficient computability means that a representation can not only describe the knowledge in the problem domain, but that there exists good implementations to solve the problems as well. A "good" implementation is defined here as one that can resolve queries to the knowledge base in a time-efficient manner. Predicate logic itself does not address the

issue of computation, because the semantics of logical expressions are defined in a purely static, declarative way. Robinson ([Robinson 65]) showed that by restricting the form of logical predicates, logic could not only be computable, but *efficiently* computable by reducing the time complexity of a brute-force search. Some efficient implementation mechanisms are described in the logic programming literature ([Cohen 85]).

A logic interpreter is a program that, given a collection of facts and a set of relationships among those facts (called *clauses*, *statements* or *rules*), determines if another fact or relationship (the *query*) can be inferred. The mechanism for performing this task is based on the resolution principle of Robinson ([Robinson 65]), and a program that implements it is a *resolution theorem prover.* Resolution theorem proving is a form of constraint satisfaction where all plan variables to be constrained are defined over a finite domain, and can (in principle) be solved by exhaustive search. For example, consider the following clauses:

1. $\forall x \in$ characters, onstage(x) $\wedge$ speaking(x) $\supset$ main_character(x).
2. onstage(Sonny).
3. onstage(Micheal).
4. onstage(Vito).
5. speaking(Sonny).

The domain of the constrained variables in this case is {Sonny, Michael, Vito}. If the query is "speaking(x)", this can be solved immediately since clause 5 fully constrains

x to the value "Sonny". If the query is "onstage(x)", x is under-constrained; it is equally likely to have the values "Sonny", "Michael" and "Vito". If the query is "main_character(x)", no one clause constrains the value of x. Instead, the unconstrained problem in clause 1 may be solved by recursively searching for the solutions to the subproblems "onstage(x)" and "speaking(x)" to arrive at "x = Sonny".

An important issue in the computability of a knowledge representation scheme is *monotonicity* ([Minsky 75]). Non-monotonicity results when a statement is added to a knowledge base and, as a result, some previously derived conclusions are no longer valid. To deal with non-monotonicity in a logical system there must be a mechanism to add the new valid facts to a knowledge base and a means to remove the old invalid ones. These are usually called "assert" and "retract".

Summary of Logic Programming Representation

Although the one best formalism for representing all forms of knowledge may never be found, there are certain minimum requirements that any knowledge representation system must meet: it must have a formal notation, it must be semantically clear (unambigous) and it must be computable without combinatorial complexity. For scene animation, logic programming meets these requirements.

Pure predicate logic may have many drawbacks in knowledge representation; for example, it cannot directly represent the concept of property inheritance. But as Hayes points out ([Hayes 77]), most of these difficulties stem from the conventional syntax of first-order predicate calculus and not from the limitations of logic *per se*. Despite the possible limitations of predicate logic for other purposes, such as natural language understanding, it is a contention of this thesis that the clarity of meaning inherent in logic programming makes it a good basis for describing the constraints on the actions of characters in a scene, particularly if the system is to interact with those lacking a background in computing. As a result, logic programming is used as the basic representational scheme of the Director's Apprentice.

Conflicting Knowledge

A great deal of research has been conducted by AI researchers into finding ways to deal with conflict and its consequence, *inconsistency* ([Hayes 74,77], [Reiter 78,80], [Doyle 79], [McCarthy 80], [Zadeh 83], [Borgida 84], [Halpern 85], [Horvitz 86]). Conflict-management techniques in AI are usually concerned with updating a knowledge base which is initially consistent, but becomes inconsistent when new information is

added. Correcting these inconsistencies is referred to as *truth maintenance* ([Doyle 79]), and remains a largely unsolved problem, although local corrections can be made by compromise techniques as discussed in [Borgida 84] and [Horvitz 86].

Having commonsense general knowledge in a system means that the system can assume facts that are not stated explicitly. Conflict arises when these assumptions are later contradicted. For example, if the system fails to satisfy the predicate "onstage(x)" , then the reasoning system may employ "failure as negation" to assume the converse *by default*, i.e. that "¬onstage(x)" is true. If "onstage(x)" *is* later inferred, the conflict must be resolved. Default reasoning (see [Reiter 80]) may also take the form of "system default" clauses, that are assumed true if not proven otherwise; this greatly reduces the number of explicit rules needed.

Another source of conflict in a knowledge representation comes from the presence of *uncertainty* . Conventional predicate logic is boolean in nature, having only the states **true** and **false**. Uncertainty is seldom boolean, requiring a continuous range of values to represent it ("somewhat sure", "fairly sure", "possible", "doubtful", "unlikely" and so on). The successful Mycin project ([Shortliffe 76]) used *certainty factors* to reason about medical diagnosis problems. Certainty factors use arithmetic operations — based loosely on Bayesian probability — that replace the "and" and "or" of conventional

logic. The ability to reason with continuous uncertainties is important in scene description, although scalar certainty factors are inadequate.

Rather than adapt predicate logic to deal with continuous domains, some have dispensed with it entirely, adopting instead an alternate logic having more than the two states "true" and "false". This is the motivation behind the use of *fuzzy logic* ([Zadeh 83]). With fuzzy logic, imprecision in the *definition* of an event is acceptable, as well as uncertainty as to its occurrence. For example, the statement "John is *fairly close* to Mary in Scene Three" is a fuzzy statement since the predicate "fairly close" is not precisely defined. The statement "the probability is 60% that John is *within one foot* of Mary in Scene Three" is not fuzzy reasoning since the predicate "within one foot" *is* precisely defined. The use of fuzzy logic for representing knowledge is controversial. Hayes([Hayes 74]) maintains that whatever advantages are provided by fuzzy reasoning are offset by losing the clear semantics of conventional (two-valued) logic.

Planning

Generating only reasonable actions between widely spaced keyframes involves *planning*. Thus, much of the AI research in planning and reasoning ([Sutherland 63], [Fikes 71], [Hewitt 72], [Shank 77], [Sacerdoti 77], [Kahn 79], [Stefik 81], [Kautz 82], [McDermott 82], [Wilensky 83], [Allen 83], [Dean 85], [Sobek 85], [Stuart 85]) is relevant to animation. Some representations of a plan of action are a set of keyframes,a play script, a "State Space" and a hierarchy of constraints.

Keyframes

An animated film can be planned as a sequence of keyframes, but this is inadequate for generating all but the most trivial sequences because the system has little knowledge of what should and should not come between the keyframes. As a result, for good animation the ratio of film frames to keyframes may be as low as ten to one[2], and at a film rate of 24 frames per second, over one hundred fully-specified keyframes may be needed to produce just one minute of animation. This may represent several hours work to the animator, and the result may be invalidated by making a small change to the scene.

---

2 From personal experience.

Play Scripts

A play script may also be the basis of an action plan if it is first transformed into some machine-understandable form[3]. Although a script usually specifies enough information to determine the major actions in the scene, there are details missing that must be filled in by the imagination and experience of the director. A line of the script may simply read "Bill exits", but before the action can be animated, some questions must be answered, including:

1. Which character is Bill?
2. Where is the exit?
3. Are there any stage props in Bill's way?
4. Does Bill walk around the stage props? If so, does he pass downstage or upstage of them? Does it matter?
5. Can (and should) Bill walk *over* any of the stage props?
6. If there are other characters in the scene, does Bill approach or keep away from each of them?
7. How fast does he move? Does he walk, run, hop, skip, jump? Does he have a limp?

---

3 The issue of automating the transformation from natural language into a suitable data structure will not be addressed in this thesis.

8. Does he make any gestures along the way? If so, what gestures are characteristic of him (puts hands in his pockets, scratches, swings his arms)?

The answers to these questions and others help to transform the script, as well as general constraints on what is reasonable in a scene and what is not, into a detailed *plan*. The key problem is to structure this plan for efficient processing and easy modification.

The "State Space" Approach

The assumption in the State Space approach (as in [Fikes 71]) is that the solution to any planning problem in the domain can be achieved by piecing together the correct sequence of operators taken from a pool of predefined steps. This is essentially an intelligent sorting problem, where each step in the solution of the plan mates with a unique "best" piece taken from the operator pool. An analogy can be made to the solution of a jigsaw puzzle, where every possible step that can be taken to solve the problem is represented by one of the pieces. A plan then consists of a sequence of correctly matched pieces.

STRIPS

A State Space plan is used in STRIPS ([Fikes 71]), and consists of a model of the present world (the starting state), a model of the desired world (the goal state), and a pool of

operators which transform one model description into another. Each operator is a list of conditions to add to the world model, a list of conditions to remove, and a "precondition" which fires the transformation if it matches the current world state. Each planning step selects an operator to best reduce the distance (or cost) from the current state to the goal state. "Cost" functions associated with each step are the only constraints on the plan as it is developed.

Limitations of the State Space Approach

A number of problems make the "State Space" approach inadequate for the planning needs of scene animation. The first problem is the need to divide up all possible plans into a number of discrete steps from which the best sequence is chosen. If the number of individual steps that may be chosen from is small — of the order of a few dozen — a step-by-step approach like this may be practical. In scene animation, however, the search space is large, as there are numerous ways of performing a scene (such as getting from one place on the stage to another). To have to select each next step from a catalogue of operators could be very slow. An error in planning at an early stage could also lead to costly backtracking.

The second problem is that the concept of the *simultaneous* action of a number of independent agents cannot be conveniently represented by a linear sequence of operators. If each operator transforms the world in terms of the action of one agent, then a different operator is needed for each of all possible combinations of actions of the multiple agents.

The third problem, common to all state-transition-based planners is the *frame problem*, which results from the need to establish what is still true after each state transition. This arises because recent state transitions may have altered the effects of previous state transitions. For example, suppose the script calls for a character to cross the stage, and then exit. After this has happened, can we reasonably assume that:

1. the stage props are still in exactly the same locations?
2. the other characters are still in exactly the same locations?
3. the other characters still interact with each other exactly as they did before?
4. the clock on the wall shows exactly the same time?

Ask a human director these questions and she might reply "yes", "maybe", "probably not", and "no", respectively. However, since State Space planners such as STRIPS are state-transition based, the answer to all of these questions would have to be "yes", because every transition is assumed to leave all states of the world unchanged *forever* un-

less specifically stated otherwise. This is called the "STRIPS Assumption" and it is a natural consequence of state-based planning.

In contrast to state-based planners, time reasoning in the Director's Apprentice is interval-based (like [McDermott 82] and [Allen 83]). That is, events are declared as being true between pairs of keyframes, and these intervals can be independently specified for different characters. As a result, the frame problem does not arise, as we can always state explicitly over what *span* of time any fact is true. A further consequence of explicit time representation is the ability to infer such time-based qualities as velocity and acceleration.

The "Hierarchical Planning" Approach

Each element in a plan *hierarchy* may represent an entire sequence of operations. Other elements may represent entire classes of sequences. These elements form a hierarchy of *abstraction levels* from the most abstract (such as "plan next scene") to the most detailed (such as "increase right hip angle by ten degrees"). In comparing hierarchical planning to the State Space approach, Sacerdoti writes ([Sacerdoti 74], p. 117):

> A superior approach to problem solving would be to search first through an *abstraction space*, a simplifying representation of the problem space in which unimportant details are ignored.

This is the approach favoured in the Director's Apprentice and in Zeltzer's Task Manager. Conventional planners search the tree of plan choices by fully exploring the first piece of the solution path before going on to the next step in that sequence. If it turns out that the next step is impossible or inappropriate, the time spent expanding the steps leading up to it have been wasted. With a strict depth-first search, such as is used in STRIPS ([Fikes 71), PLANNER ([Hewitt 72]) or PROLOG ([Clocksin 81]), the result can be a combinatorial explosion of time, making plan calculation impractically slow. In contrast, a hierarchical planner such as NOAH ([Sacerdoti 77]), MOLGEN ([Stefik 81]) — or the Director's Apprentice — roughs out the entire plan from start to finish before filling in any details of how that plan is to be achieved. Sacerdoti writes ([Sacerdoti 74], p. 121):

> This search strategy might be termed a "length-first" search. It pushes the planning process in each abstraction space all the way to the original goal state before beginning to plan in a lower space.

A hierarchical planning approach defers constraint decisions until after the broad generalities of the plan have been worked out. This prevents early decisions from committing the plan prematurely to a dead-end course . As Dean ([Dean 85]) describes:

> Given that there exists no strong warrant for choosing one plan over another why should a planner make a choice at all? In certain situations it would seem that procrastination is appropriate. The problem was that by committing early (and arbitrarily) the planner might have to explore a large number of alternative orderings before finding one that worked.

This process of solving high-level goals before detail goals, comparable to "top-down" programming techniques, continues until all of the constraints are solved. Sacerdoti ([Sacerdoti 74] p. 133) describes the process:

> It is desired that the system's planning efforts focus on reasoning about states of the work that are likely to be traversed in the course of robot execution. Thus, the overall planning should be roughed out in an abstraction space that ignores enough levels of detail so that the rough plan is fairly certain to succeed.
>
> The process of alternatively adding detailed steps to the plan and then actually executing some steps can continue until the goal is achieved.

Example of Hierarchical Scene Planning

In Figure 2-6, characters A and B are to leave the scene via one of two exits.

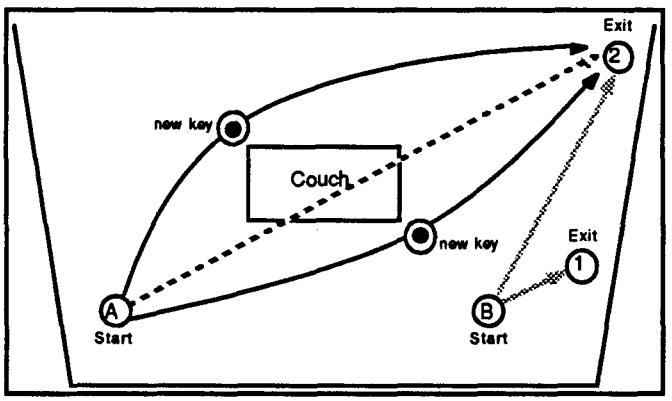

Figure 2-6 Exit Choices
Characters A and B
are to leave by one of the two exits.

Some of the constraints on Character A as he exits include having to choose the best exit, not colliding with the couch and not interfering with character B. A partial plan hierarchy that might be generated for this scene is shown in Figure 2-7.

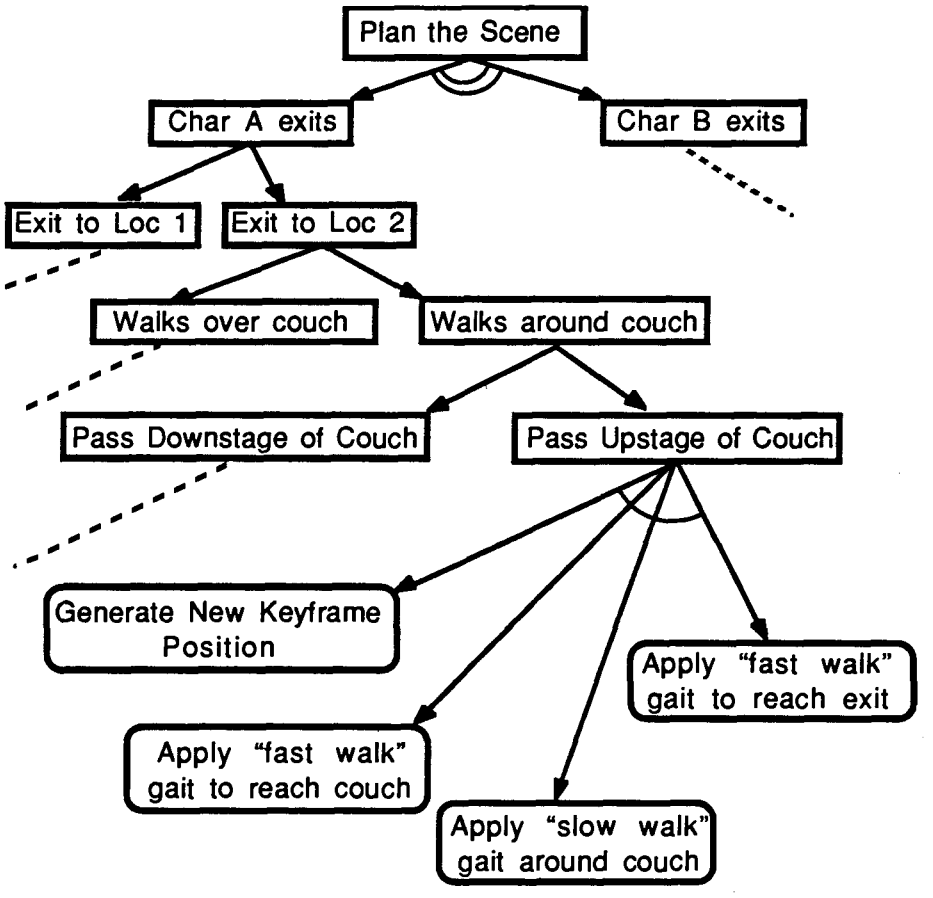

Figure 2-7 Hierarchical plan organization: each deeper level represents a new constraint

(In this diagram, a single arc indicates a sequence of actions, while a double arc suggests possible parallel actions. All other child-nodes are 'or'-selections. Dotted lines indicate other solution paths not explored. Terminal nodes, with rounded corners, indicate primitive actions).

As the tree of choices is traversed, new constraints are added to the plan. The result for character A is shown in Figure 2-8.

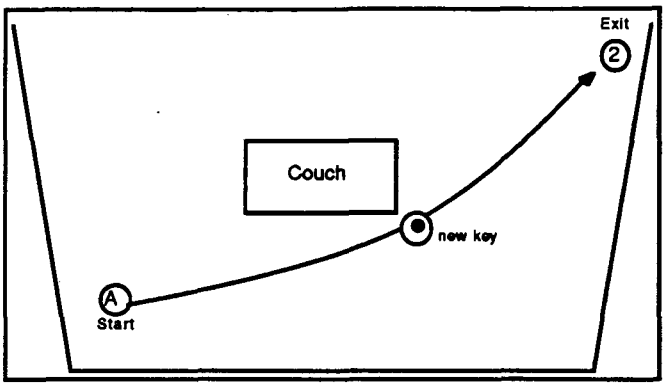

Figure 2-8 Plan solution
Solution to problem in Figure 2-6 for character A

Improvements to Hierarchical Planners

Hierarchical planning can be strengthened by providing an explicit representation of the plan structure as a "procedural net" ([Sacerdoti 77]). Such a network description contains both the procedural and declarative knowledge in the plan. The procedural knowledge expands higher abstraction goals into more detailed subgoals. The declarative knowledge consists of *critics* (rather like demons) that checks for, and corrects, any conflicts that result from subgoal expansion.

A paper by Christopher Stuart ([Stuart 85]) describes a system that extends the idea of hierarchical planning to multiple agents, using synchronizing primitives similar to those found in operating systems for resource sharing.

The Find-Path Problem

"Find-path" is a problem from robotics research ([Brooks 83], [Lozano-Perez 83]), where an autonomous robot must find the shortest route from one place to another across a cluttered work area. Brooks' solution ([Brooks 83], p. 190), is to "shrink the object to a point while at the same time expanding the obstacles to the shape of the moving object".

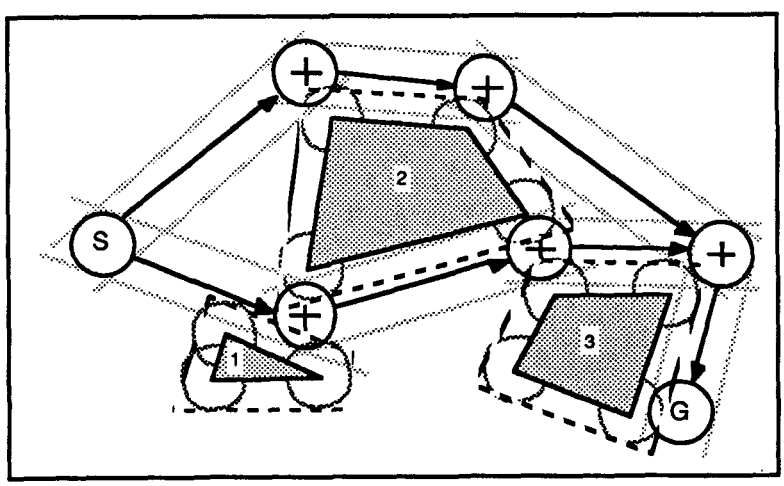

Figure 2-9 A find-path solution

As illustrated in Figure 2-9, a character (typically a mobile robot, but could also be human) is moving from the starting location "S" to a goal "G" past obstacles 1, 2 and

3. By effectively expanding the obstacles by the radius of the character, the worst-case size of the obstacles can be determined, and, by reducing the character to its centre point (marked with a cross), any route[4] to the goal that does not intersect these expanded obstacles is a find-path solution. Some algorithms for dealing with rotatable (non-circular) characters have also been proposed ([Brooks 83] and [Lozano-Perez 83]), but tend to be much slower. In scene animation, rotation is not an important issue, as humans are fairly circular when viewed from above.

The find-path problem occurs in most scene animation situations, and is sufficiently critical that it should be placed at the top of any planning hierarchy. But, the nature of find-path in scenes is different from that found in robotics problems; in robotics, typically the only constraint on the action of the robot (apart from collision avoidance) is to minimize the distance travelled so as to minimize time. When humans move from place to place on a stage set, there are many other subtle influences on their actions, and find-path is only one level of a large *hierarchy* of constraints that must be accommodated and compromised.

---

4 A route is a sequence of connected straight lines joining the start to the goal.

Temporal Logics

Many AI workers in natural-language research have studied the concept of time as found in stories. Key papers include [Allen 83] and [McDermott 82]. The basic intent of such research appears to be story *understanding*; a system is expected to digest a natural-language version of a story containing references to time (such as "before", "after", "later", "sooner" and so on), and then to be able to answer questions of the form "who did what, when?".

While this research may be helpful in the future, particularly for transforming script text into machine-understandable form, it appears, at this time, to be too preliminary to be useful in scene animation.

Chapter Summary

Of the attempts that have been made to improve the power of figure animation systems, the work of David Zeltzer is closest in its aims to the Director's Apprentice, although the high-level task manager was not completely implemented. And while systems like that

of Girard, and Korein and Badler are well suited to animating generalized limb structures, they need to be adapted to the particular constraints of human figures.

A good representation for knowledge should be formal and semantically clear. Predicate logic has these qualities. A good representation for scene animation also needs to be able to deal with the continuous-valued world of distances and forces for which logic programming requires extensions.

Planning methods designed for blocks-assembly tasks, such as STRIPS, are not powerful enough in themselves for scene animation purposes. Hierarchical designs are more useful, since the scene planning space, that is, the range of options available to the character, is large.

Bringing these elements together into a cohesive whole was one of the aims of the Director's Apprentice project. The extent to which this has been achieved is discussed in Chapter 4.

Chapter 3:
Satisfying Scene Constraints

Many statements present in the knowledge base provide *constraints* on how the characters are expected to behave. Not all constraints are applicable to every character in every scene; so they are defined *conditionally*, requiring an *inference procedure* to solve them. Also, since the statements are entered independently, they may conflict, and cannot all be satisfied simultaneously.

In scene-level animation, the system develops a *plan* for each of the characters, and the current state of the simulated world (locations, sizes, relationships, etc.) is expressed by the values of a set of *plan variables.* A *constraint* associates with each plan variable a set of acceptable values, which may change over time.

Operations on Constraints

A paper by Mark Stefik ([Stefik 81]) identifies three fundamental operations performed on a constraint : it is *formulated*, then it is *propagated*, and finally it is *satisfied.*

Constraint Formulation

Constraints are formulated when new commitments about the characters are made by the system. In scene animation these commitments may be to keep characters on the stage, to prevent characters from walking into props, to maintain certain gait patterns for each character and to keep antagonistic characters away from each other. Some constraints are of a general nature, and they are applicable to many situations; for example, "keep all characters on the stage floor". Other constraints are more situation-specific, such as "character x walks *around* couches but climbs *over* fences. Solving general constraints first may save time: how to climb a fence is not an issue if the character never comes near one.

In conventional constraint satisfaction systems ([Sutherland 63], [Borning 79], [Seidel 81]), the number and kind of constraints are all fixed and known at the outset. For example, a paper by Raimund Seidel ([Seidel 81]) describes a systematic method for solving binary (two variable) constraints. While this method may be quite efficient, having all constraints fixed unconditionally at the outset is too limiting for a scene having many characters.

Constraint Propagation

Constraints are not all alike: some are more general than others, some easier to satisfy than others. Putting the constraints in order and using the solution of each to help solve others is *constraint propagation.* Usually the narrowest range (fewest options) are solved first. For example, the constraint "the character is onstage" has a narrow range since it is either **true** or **false.** Harder (wider range) subproblems such as "the character is in a position of strength" are either solved — or skipped — by using the results of the easier subproblems to limit their range. (If he is not onstage, "strength" is meaningless.)

For another example, the constraint:

(C1) "the character is downstage from the couch"

does not constrain the character's location to any particular place; its range is infinite. In Figure 3-1, any location that falls outside of the shaded area will satisfy constraint C-1.

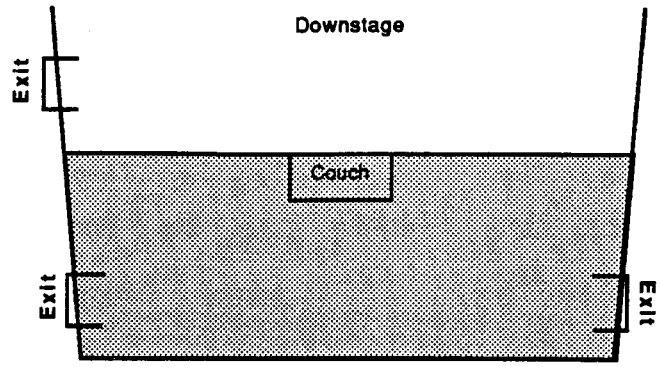

Figure 3-1 Underconstraint Space
Any location within the shaded area is a solution for Constraint C1

If a narrower range subproblem is found in the knowledge base and solved first, such as:

(C2) "the character is standing in the doorway of one of the three exits"

large branches of the decision space for C1 can be pruned away, as shown in Figure 3-2.

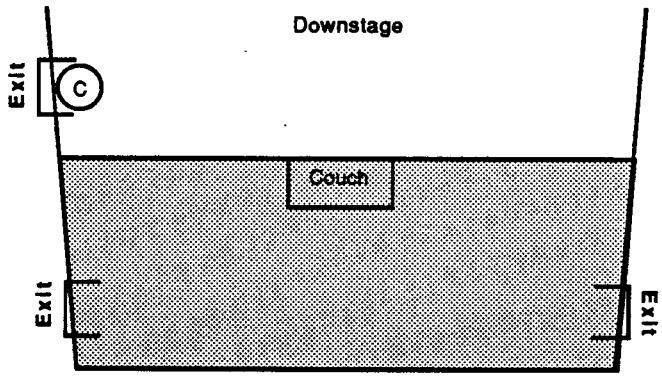

Figure 3-2 Fully Constrained Space
Constraint C1 is fully constrained after propagating Constraint C2

If Constraint C1 had been solved before solving C2, the system would be forced to test every location in the unshaded area (i.e., the set of locations that satisfy C1) to see if it

also satisfied C2. By solving C2 first, and *propagating* its result to C1, there are only three places that must be tested for satisfaction (by C1). In Figure 3-2, only one of the exits lies within the acceptable area for C1 at downstage left, so it is the solution.

Constraint Satisfaction

Constraint satisfaction is the process of finding values for the plan variables that *simultaneously* solve all of the constraint conditions. By organizing the constraints into categories from most general to the most detailed, and from the narrowest range to the widest range within each category, an efficient scheme of constraint satisfaction results. The generality hierarchy (called an "abstraction hierarchy" by Stefik [Stefik 81]) minimizes backtracking over needless details while the range hierarchy forces relatively easy subproblems to be solved first.

A problem occurs when there are loops in the constraint hierarchy; this commonly arises when solving systems of equations which are sometimes solved by numerical techniques such as *relaxation*. This is an iterative technique involving a guess, the estimation of the error, and a new guess based on whether the error is improving or getting worse. This process continues until the error is acceptably small, or the relaxation

fails to converge. One common implementation of relaxation between interacting constraints is to only solve each conflicting constraint *partly* (make a small incremental change), to minimize the interactions between the changes. These incremental changes are repeated until stability is reached.

Continuous-valued constraints involving inequalities are particularly hard to solve, since satisfying the inequality conditions is generally inadequate. Suppose a scene to be animated includes a race between a hare and a tortoise. A constraint that may be found in the knowledge base is:

(1) speed(hare) > speed(tortoise)

When this problem is solved using relaxation, a first guess at the constraint solution may be

(1) speed(hare) = 10^{10} m.p.h.; and

(2) speed(tortoise) = 0 m.p.h.

At this point, the inequality has already been satisfied, but this solution is clearly unacceptable. The problem is to determine what to make for a second guess, since the first guess already satisfies the constraint. One approach is to provide a probability density

function associating a likelihood with each speed value, as illustrated in Figure 3-3. By stepping away from the expected value (point of maximum probability) in small steps, a sequence of decreasingly likely values for the speed of each character is obtained. Thus it is the *likelihood* of a speed value that is relaxed, rather than the speed itself.

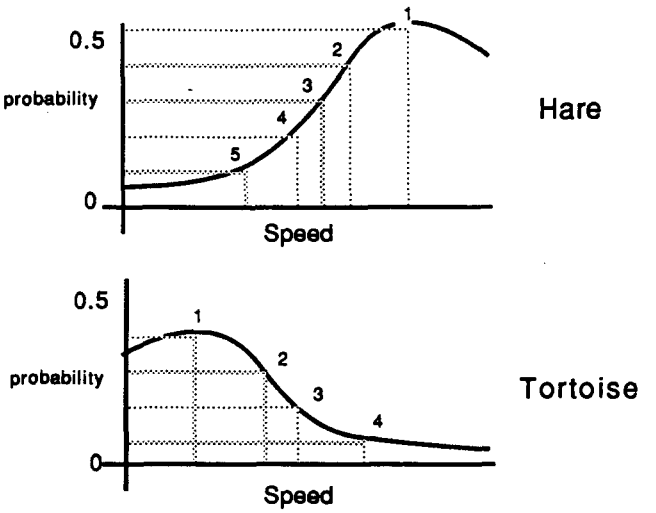

Figure 3-3 Probability Relaxation
Probability associated with each speed value for the hare and the tortoise

ThingLab

In 1979 Alan Borning published a description of a constraint-based simulation laboratory called "ThingLab" ([Borning 79]). This SmallTalk-based system allowed the user to specify (for each constraint) an invocation rule, an error measurement test and explicit methods for reducing the error.

ThingLab has a number of powerful features. It has both propagated and relaxed constraints; it has hierarchically-defined data types allowing for inheritance classes of constraints; and it has the convenient graphical user interface common to all SmallTalk-based systems. However, ThingLab has no facility for performing general inference, so there is no means of saying "under these circumstances, use these constraints"; as a result, all constraint satisfaction methods defined in the system must always work under all circumstances. Since inference is essential for implementing *conditional* constraints set, the ThingLab design is of limited use in animation.

MOLGEN

Another paper ([Stefik 81]) describes a system for constraint satisfaction that, like Borning's ThingLab, grew out of the author's Ph.D. research. Stefik's system, called "MOLGEN", is designed for planning a sequence of operations to construct an experiment in molecular biology known as "gene splicing". The constraints defined on these operations consist of a set of elimination rules. Genetic selections that do not meet the current set of criteria are pared down until a set of bacterial organisms is left that meets the requirements of the experiment. MOLGEN uses a form of hierarchical planning where easier problems (those with few constraints) are solved first, and the partial results produced are used to solve the harder problems. This bottom-up propagation continues

until all of the hardest problems are solved (are fully constrained).

The concept of propagating the results of easier problems to help solve harder ones is formalized in a paper by Dechter and Pearl ([Dechter 85]). It shows how such propagation can be used to reduce backtracking, by preventing the attempted solution of problems whose results are not needed.

Constraint Propagation in Scene Animation

The wider the range of values that *could* satisfy a constraint, the harder will be the constraint satisfaction problem. For example,

C1: "*a* is true"

is relatively easy to satisfy since there are only two possible values to be tested for boolean variable *a*.

C2: "$\exists n \in \{1,2,3,4\}$ p(n)"

is harder, since p(n) may have to be tested for all four values of n. Still harder is

C3: "$\exists x$, -1.0 < x <+1.0, q(x)"

where x is a continuous (real) variable, since there are an infinite number of test values for q(x).

In the Director's Apprentice, a form of logic programming ([Clocksin 81]) — which can be thought of as constraint satisfaction restricted to finite sets as in C1 and C2 above — is used to guide the process of problem reduction, preventing the attempted satisfaction of constraints that are irrelevant to the scene at hand. This is essential since the total number of constraints that *could* be applied to a scene is enormous. Only those continuous-valued constraint problems that are needed in any given scene are, in fact, solved. This contrasts with conventional constraint systems in which all continuous-valued constraints are solved. Discrete-valued constraints are propagated and satisfied by the efficient mechanism of resolution, and the results decide which (harder) continuous-valued subproblems will be attempted. Constraint solving systems that lack general inference cannot conveniently do this.

In summary, constraint satisfaction in scenes is complicated by the need for both discrete and continuous-valued constraint satisfaction in the same system. The former can be handled efficiently by logic programming, but the latter requires numerical techniques, since exhaustive search cannot be applied over an infinite domain. The challenge is to integrate these two approaches into a hierarchical system of satisfaction techniques.

Sources of Constraint in Animated Scenes

The script states or implies goals for each character, and many conflicts arise because of inconsistencies between the influences on the way a character should move between keyframes. For example, interpersonal relations (x *loves* y, z *is afraid of* w, etc.) can be consistent with some goals but can conflict with others. Characters can also interact with stage props, avoiding dangerous objects and drawing close to attractive ones. These tendencies are often emotional in nature.

Aaron Sloman's paper ([Sloman 81]), entitled "Why Robots Will Have Emotions", discusses what is required to represent the motivation of a robot imitating human behavior. While the feasibility — or the need — of robots behaving like people is questionable, animated figures that represent people have to display convincing human-like behavior, and as a result, some of his observations are relevant here.

First, a character's environment may change unexpectedly, and the simulator should allow for this. Sloman writes:

> The environment is not static: opportunities and dangers, may all vary from time to time. Changes will need to be perceived or predicted. Predictions will not always be re-

> liable. This implies a need for constant monitoring, and ability to notice and deal with the unexpected.

Second, some of a character's motivations may be at odds with his plans. Should the character follow a spur-of-the-moment impulse, or stick to his major goal? Sloman writes:

> There may be intense, though less important, motives which conflict with a long term goal. It is necessary to be able to interleave pursuit of different intentions.

Third, it may not be possible to satisfy all of a character's goals simultaneously. In Sloman's words:

> Different motives in the same individual may be inconsistent. Mechanisms and strategies for dealing with inconsistencies will be needed.

Conflict and Planning

The Director's Apprentice is not intended as a stand-alone animation system that interprets scenes automatically, as this would limit it to trivial or highly abstract scenes as in the work of Kahn [Kahn 79]. Instead, its aim is to allow an animator to spread manually entered keyframes further apart while the system continues to generate believable

action. The "script" used contains the original action of the play[1] but may also contain any interactively-added annotations and alterations that the director feels are needed to achieve the desired effect.

Thus the user, while specifying as few keyframes as possible, is a *partner* in the animation process. The computer system serves as a planner that fills in the gaps left by the animator. As in any planning problem there is an initial state (the starting keyframe), a final state (some later keyframe) and a set of constraints that limit the space of choices available for transforming the initial state to the final. It is the nature and handling of these constraints that is of interest here.

Constraint Satisfaction Examples

Suppose the script called for the character (Character 1) to exit the scene from Upstage-Left through a doorway Downstage-Right. Figure 3-4 illustrates this. If this were the only constraint on the character, any figure animation system capable of gen-

---

1 The play script is assumed to have been translated into some machine-understandable form.

erating a straight walk could handle it.

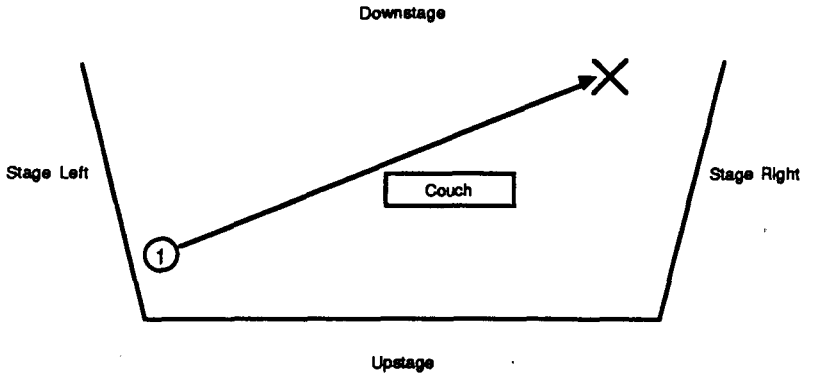

Figure 3-4 Character exits to right unobstructed

Suppose now that the animation director wishes to experiment with the furniture placement. She moves the couch downstage in such a way that the character's straight-line path intersects it. (This may happen accidentally.)

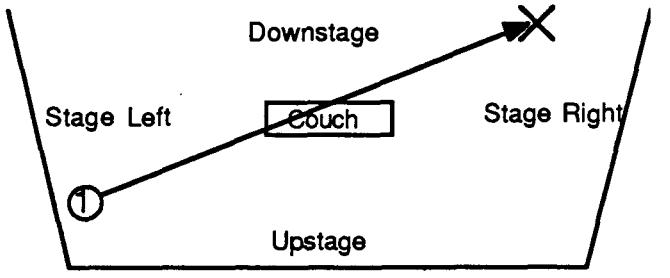

Figure 3-5 Character path now intersects couch

In order to animate this scene in a reasonable way, the system's general knowledge must contain the *discrete* constraint

$$\forall_f \in \text{frames},\ \forall_c \in \text{characters},\ \forall_p \in \text{stage-props}:$$
$$\text{Space}_{c,f} \cap \text{Space}_{p,f} = \varnothing$$

which says that for every character and stage prop in the scene, the space occupied must not intersect that of any other character or prop in any frame.

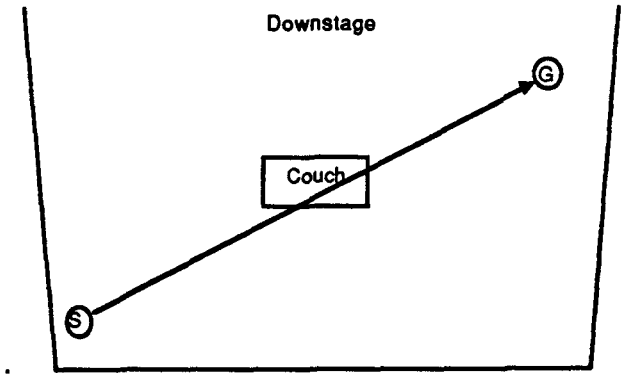

Figure 3-6 No Constraints
Figure walks from Point S to Point G, ignoring constraints

The system will generate a new keyframe between S and G such that the resulting two-part path satisfies all of the character's constraints. The location of the new point will be equidistant between Point S and Point G, although any other subdivision scheme would work just as well. Figure 3-7 illustrates a selection of the possible locations for the new intermediate keyframe point.

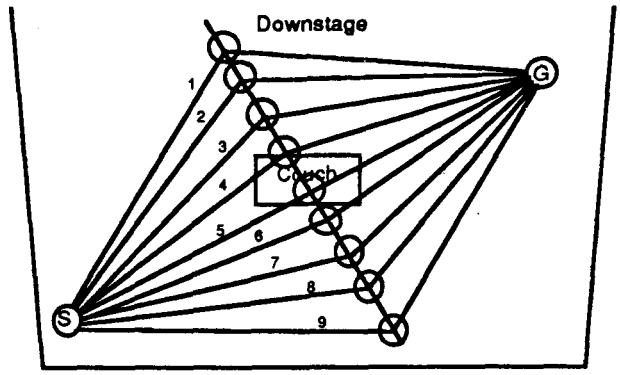

Figure 3-7 Nine possible choices for the new keypoint

The next issue to be resolved is which set of constraints will influence the movement of the character as he walks from Point S to Point G. Specific facts taken from the script are combined logically with general principles in the knowledge base to produce a *constraint profile* that is appropriate to the character in this scene. Table 2-1 lists five.

	Constraint	Type	# of Solutions
1	**If** Frame = *n* **then** character is speaking	Discrete	1
2	**If** character is speaking **then** pass downstage of obstacle	Continuous	∞
3	**If** object is couch **then** obstacle is present	Discrete	1
4	**If** obstruction is present **then** avoid obstacle	Continuous	∞
5	**If** walking **then** take shortest path around obstacle	Continuous	∞

Table 2-1
Five interacting constraints on Figure 3-7

Here there are two discrete constraints on the action of the character (1 and 3), each

having one solution (either **true** or **false**). At the same time there are three continuous-valued constraints, each having an infinite number of solutions (2,4 and 5). Since the discrete constraints have only two possible values each, they divide the total solution space into four regions. Picking the right combination may cut off much of the work needed to satisfy the continuous-valued problems. For instance, if the value of constraint 3 is **false**, then dealing with obstructions (constraints 2,4 and 5) is eliminated. For the purpose of this example, assume that the value of constraints 1 and 3 are **true**.

Next, the continuous-valued subproblems are addressed. Each one effectively delimits areas of the stage that conform to the rule involved. Rule 2 (which is only used since the value of Rule 3 is **true**) delimits the stage as in Figure 3-8. This effectively eliminates Paths 6, 7, 8 and 9 from consideration.

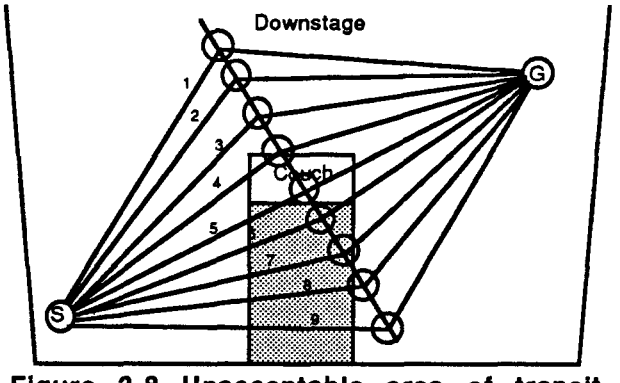

Figure 3-8 Unacceptable area of transit is shaded

Rule 4 delimits the stage as in Figure 3-9, eliminating Paths 4, 5 and 6 from consideration. This leaves us with Rule 5 to resolve Paths 1, 2 and 3. It is used last, as its range is the entire floor and it is thus the least constrained.

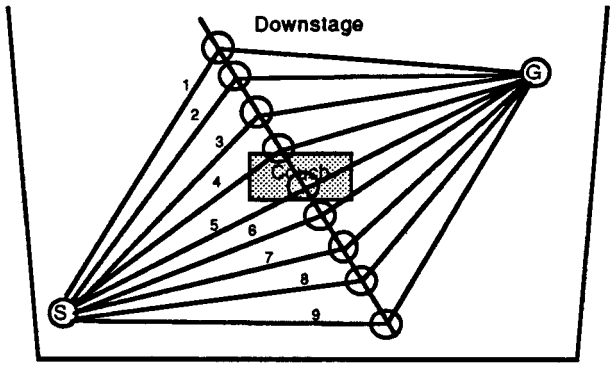

Figure 3-9 Obstruction-free area of stage

Rule 5 is special. Rather than specifying an acceptable area of the stage, Rule 5 ranks all locations with a "goodness" measure corresponding to the total path length between S and G through the new keypoint. Since the total distance for Path 3 is less than Path 2 which is less than Path 1, Path 3 is the clear winner.

Constraint Satisfaction in Path Planning

In the current implementation of the Director's Apprentice, the principal application of constraint satisfaction is in path planning, that is, deciding how a line such as "Bill walks to the bookcase" or "Mary exits stage right" should be interpreted. Some aspects

of this problem, such as obstacle avoidance, are discussed in Chapter 4. What follows is a discussion of the satisfaction of *tendency* constraints, which influence the path of a character between two keyframe positions. In Figure 3-10, the script calls for the character to moves from key position 1 to key position 2. A new keyframe position (3) is generated between them. Figure 3-10 suggests that there are four other objects in the scene ("a", "b", "c", "d"), that are attracting the character away from the normal straight path. Since these tendencies conflict with each other, conventional monotonic logic cannot resolve the character's correct position.

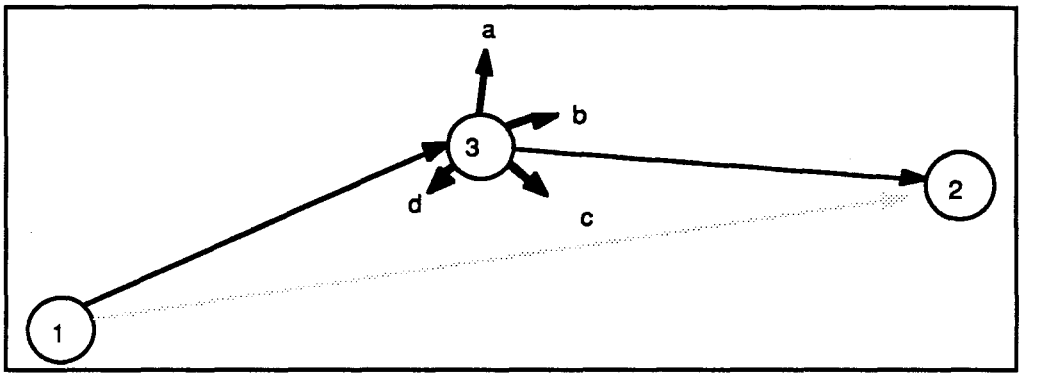

Figure 3-10 Inserted midpoint keyframe displaced by tendencies

Furthermore, since the tendency forces are time and character dependency, considerable complex inference may be required to determine the appropriate displacement functions to use. As a conclusion regarding a tendency force is reached, that value is added to the compromise list associated with that character and time.

Figures 3-11 through 3-16 illustrate how hierarchical planning with tendencies is implemented. In Figure 3-11, the script supplies the locations of keypoint 1 and 2 in Figure 3-10.

The constraint hierarchy (illustrated at right) represents the assertion that the location of keypoint 3 is:

"(Distance(1,2) >MinDist) ⊃ Compromise(midpoint(1,2),a,b,c,d); NULL"

or, if the distance between points 1 and 2 is above threshold (preventing endless subdivision) then locate new keypoint 3 by finding a compromise between the midpoint of line between points 1 and 2 and the four tendencies acting on it.

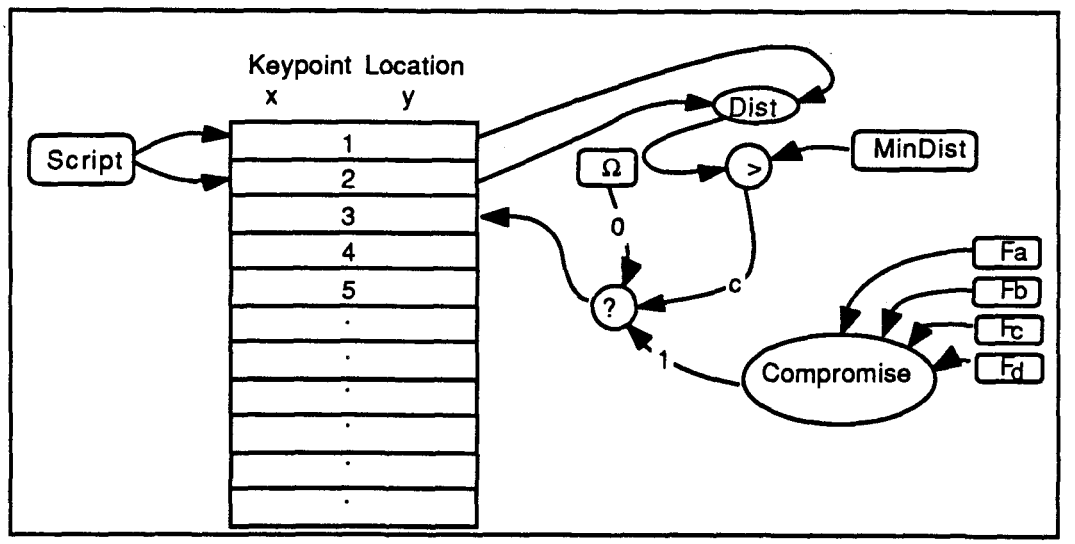

Figure 3-11 First subdivision Constraint satisfaction hierarchy

In Figures 3-11, 3-14 and 3-16 this symbol represents a decision (backtrack) point:

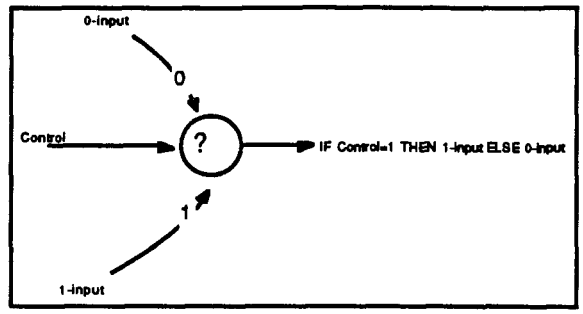

Figure 3-12 Decision point

Note that if the "minimum distance" constraint fails, the (more difficult) compromise constraint will be skipped. Figure 3-13 though 3-16 show similar constraints for two more subdivisions.

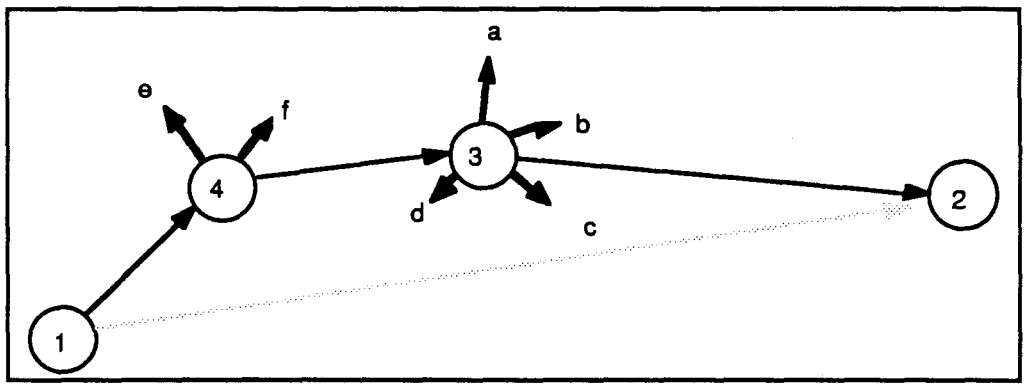

Figure 3-13 Insertion of Point 4

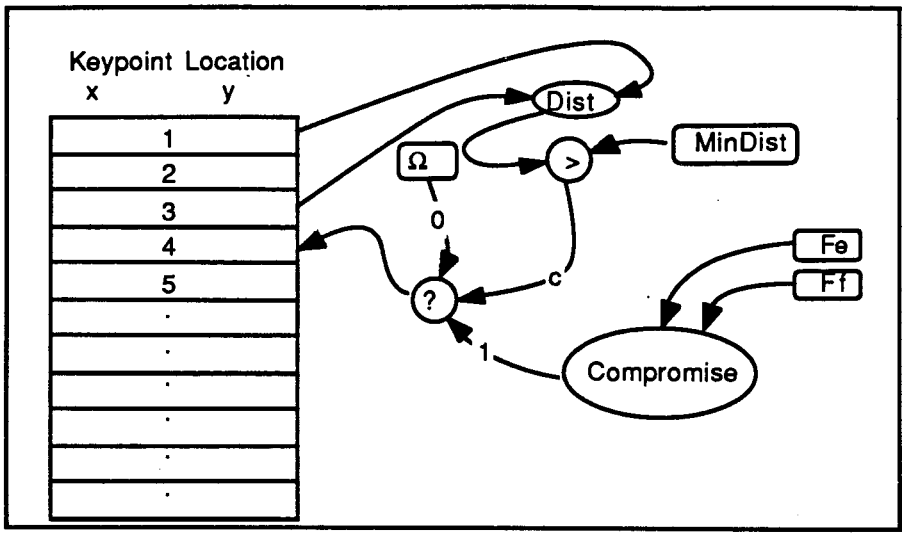

Figure 3-14 Constraint hierarchy for Point 4

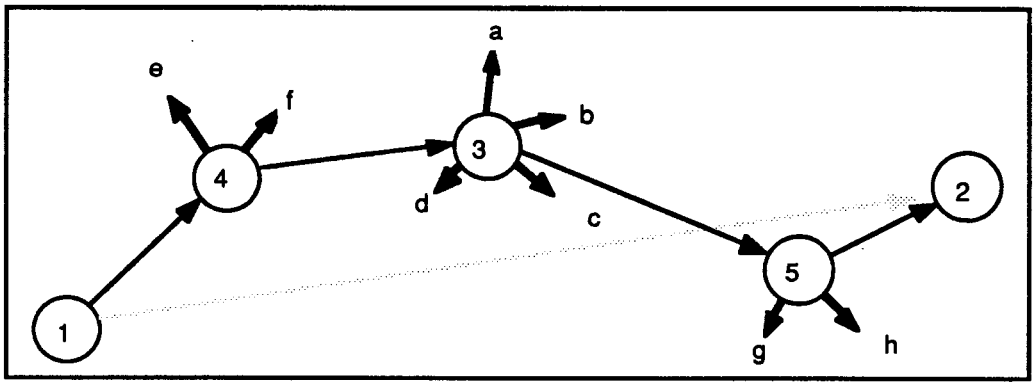

Figure 3-15 Insertion of Point 5

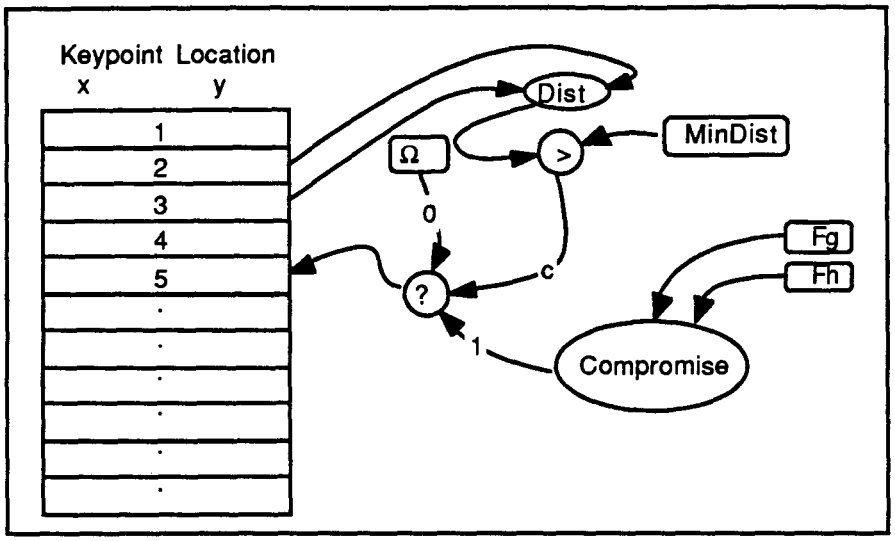

Figure 3-16 Hierarchy for point 5

In practise, a new keypoint for a particular character is only perturbed by a small amount from its central position. The whole process is repeated for each character. At each correction, the forces are calculated on the basis of where the fixed objects are, and where each character will be at that time, including the new keypoints. This series of incremental changes for each character is repeated until all of the tendencies are satisfied.

Summary

In representing knowledge for scene animation, being able to handle knowledge that disagrees is very important since so many of the influences on human motion conflict. Also,

while a lot of work has been done in AI toward developing systems that tolerate inconsistency, the needs of an animation system are different from those of the typical AI application. In this chapter it was argued that animation of scenes is fundamentally a constraint-satisfaction problem, and some characteristics of an effective planner for scenes were described.

Chapter 4:
Implementing a Scene Representation System

This chapter discusses the design and implementation of the Director's Apprentice.

The first section examines the role of the Director's Apprentice as an animation "expert system". The second section discusses how logic programming is extended for this purpose in the Scene Constraint Language. The third section describes the mechanisms behind the hierarchical planner and the scene constraint interpreter. The fourth section describes how clear paths for character action may be generated, and the fifth section gives an example of how a scene may be animated by the Director's Apprentice. The final section describes some details and some limitations of the current implementation.

An Animator's Expert System

One long-term aim of the Director's Apprentice project is to produce an "expert system" to aid animators, much as rule-based expert systems are beginning to be used as decision-support tools in machine maintenance and financial management ([Duda 84],

[Winston 84]). Of the many programs referred to by the term "expert system", some are concerned with the diagnosis of disease([Shortliffe 76], [Buchanan 84]), with mineral identification from geologic data([Duda 84]), with computer system configuration([McDermott 81]) and many other applications (see reviews in [Nau 82], [Barr 81], [Hayes-Roth 83]). Despite the fact that the term "expert system" has no widely-agreed-upon definition, all of these projects have one aim in common: to amplify human skills in areas that normally involve the judgement of experienced people. In the Director's Apprentice, these experienced people are animators and theater directors.

Existing expert systems have widely different implementations, including backward-chaining inference with certainty factors([Shortliffe 76]), forward chaining production rules([Forgy 77]) and multiple communicating knowledge sources([Erman 80]). In each case, *declarative* knowledge is strictly separated from *procedural* knowledge. Declarative knowledge in an expert system consists of two sets of facts: general principles common to all problems in the domain, and specific facts for the solution of a particular problem. The first consists of rules for commonsense "default reasoning" (such as "no character in any scene should walk through any solid object"), while the second set of facts comes directly from the script (such as "the character Carlo prefers to avoid the character Sonny"). Note that "script" is used here to mean the written text of a play, rather than a template for common action sequences (in the AI sense of [Shank

77]).

The scene-specific "Script Fact Collection" is entered and maintained by the animators. The commonsense General Knowledge Library is maintained partly by animators and partly by logic programmers. Together, the two sets of knowledge, general and scene-specific, make up the system *knowledge base.*

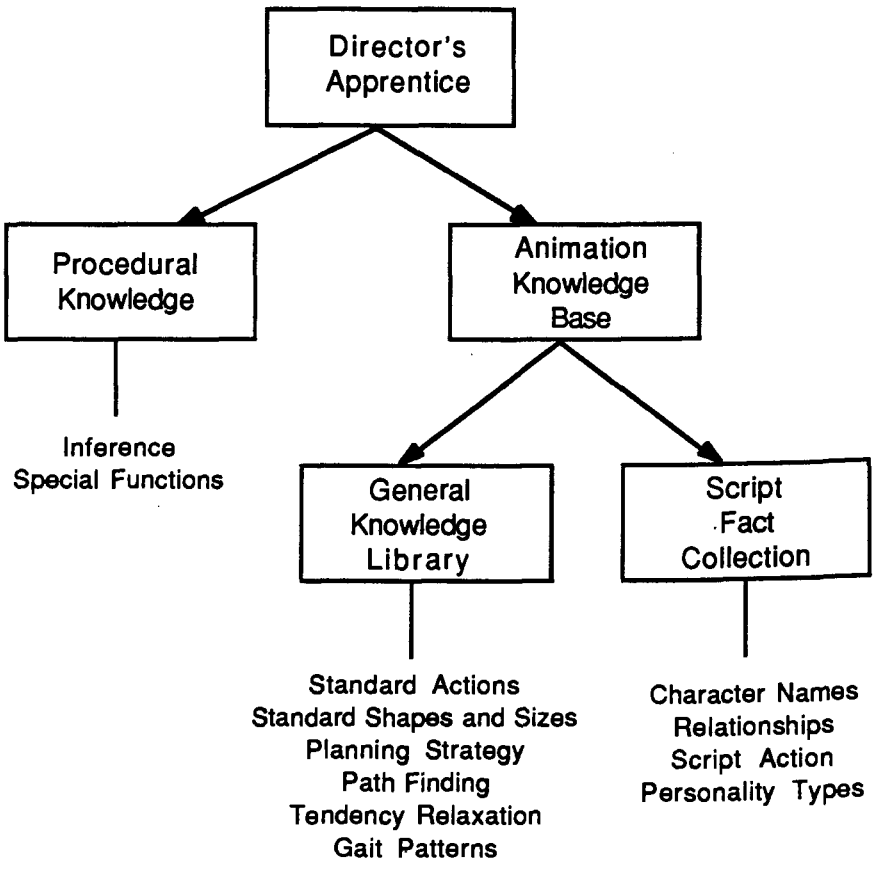

Figure 4-1 Structure of the Director's Apprentice

The procedural knowledge in the Director's Apprentice includes a backward-chaining resolution theorem prover with extensions that enable it to reason about continuous-valued constraints. This is kept strictly separate from a declarative knowledge-base of commonsense defaults and scene-specific facts. The result is only the *nucleus* of an expert system for human scene animation since a true expert system requires a user interface that not only hides the procedural knowledge (implemented), but also obscures from the user the details of the general knowledge (not implemented).

Procedural knowledge consists of the knowledge needed to infer one set of facts from another set of facts. In the Director's Apprentice, most of the procedural knowledge is incorporated into a *logic interpreter.*

Extending Logic Programming for Scene Animation

Conventional constraint satisfaction systems are concerned only with satisfying a single type of constraint: e.g., PROLOG([Clocksin 81]) with discrete variables, SKETCHPAD ([Sutherland 63]) with continuous variables. A contribution of the Director's Apprentice is to integrate both into a hierarchical planning environment.

Constraint satisfaction for continuous-valued variables within a logic programming framework may be provided by clause types that automatically resolve conflicting continuous-valued assertions. For example, the clause:

expression ⊃ (assert "Sonny" "location-is" (10.2 20.7) 6).

adds to the (scene-specific) knowledge base a fact constraining character "Sonny" to location "(10.2 20.7)" on the stage at keyframe 6 (if *expression*=**true**). If followed by the clause

expression ⊃ (assert "Sonny" "location-is" (20.1 30.5) 6 "revise")

the constraint on the location of character "Sonny" is changed to be "(20.1 30.5)". However, if the second clause instead reads:

expression ⊃ (assert "Sonny" "location-is" (20.1 30.5) 6 "compromise")

then the value "(20.1 30.5)" is added to the *compromise list* associated with the "location" for Sonny in the current context list. The compromise list is processed only when all of the other influences on Sonny's location have been accounted for. Continuous-valued assertions such as these are treated as any other in the knowledge base, and may be freely combined with ordinary clauses. This allows complex *conditional* constraints to be defined.

For example, *tendencies* that alter the path of a character between keyframes may be

likened to forces that conditionally constrain a character "c" to move in a particular direction. Figure 4-2 shows three tendency vectors, each one attracting the character to object 1, 2 or 3. Since these vectors represent three conflicting tendencies, a compromise must be found to combine them into one displacement vector representing the character's net movement tendency.

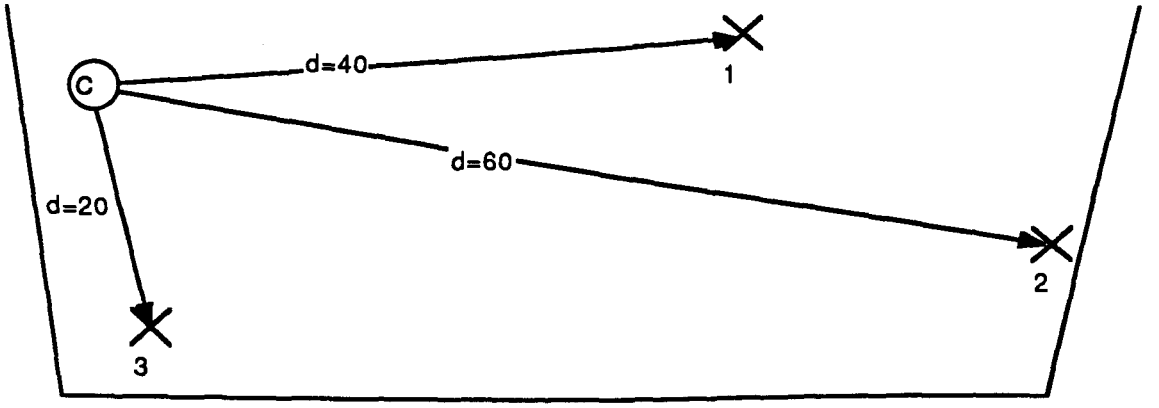

Figure 4-2 Tendencies to be attracted to three objects at different distances

Each tendency is defined as a *probability density function* relating the distance (between the character and the source) to the expected displacement of the character. Figure 4-4 shows a possible displacement function for the tendency vectors in Figure 4-2. (Each tendency may have a separate function).

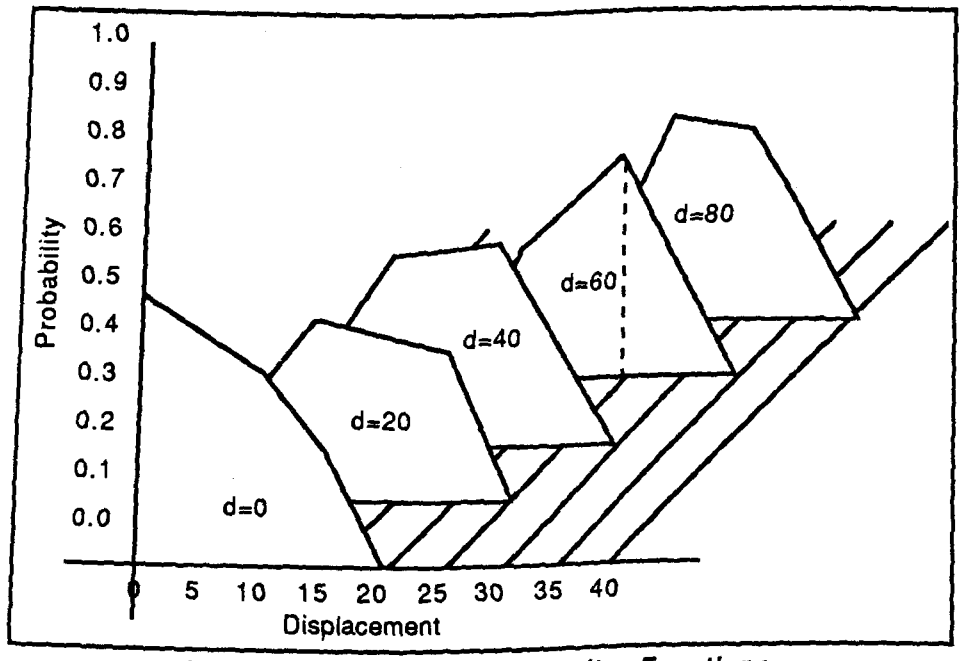

Figure 4-3 Probability Density Functions For Various Distances Between Character and Source

To determine the figure's net movement, the expected value (highest peak) of the appropriate density function for each tendency is taken. This returns a displacement value which is used as the length of each tendency vector. (In Figure 4-3, a character-to source distance of 60 returns an expected value of 25 for the magnitude of the second tendency in Figure 4-2.) Once a likely displacement value (length) has been selected for each tendency, the tendency vectors are added geometrically. If the sum turns out to conflict unacceptably with the other scene constraints, a second try can be made using a lower-probability displacement.

The result is shown in Figure 4-4 where the character is displaced towards upstage-right by the sum of the three tendency vectors.

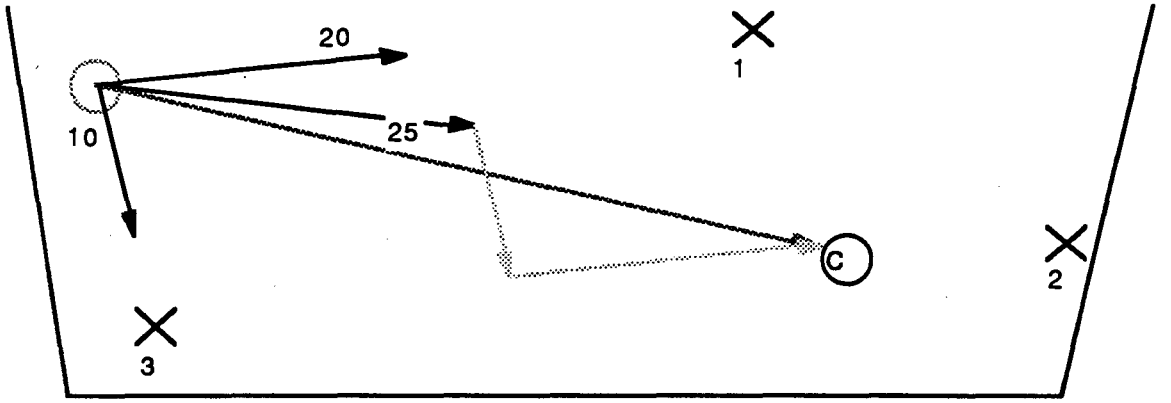

Figure 4-4 Net Movement Tendency

The Scene Constraint Language

This section gives an overview of the the Scene Constraint Language ("SCL") developed for writing knowledge bases in the Director's Apprentice. A detailed specification of its syntax is given in the Appendix.

Each SCL knowledge base is divided into general (default) knowledge, applicable to all scenes, and specific knowledge derived from a particular script. Each of these consists of a collection of data definitions, followed by a list of rules. The following table lists some

representative examples.

Type	Data Definitions	Rules
General Defaults	definitions of relationships, definitions of personality-types, sizes of standard props, area of stage	plan hierarchy, find-path techniques, tendency relaxation, gait patterns for personalities, gesture patterns for personalities
Scene-Specific	names of characters, names of relationships, locations of props, locations of characters at particular times, personality-types for each character	any special actions that overrule general defaults

Each rule consists of a name, a list of local variables, a sequence of conjunctive clauses and a conclusion. For example, these clauses define the stage rule "don't pass downstage of the main character":

onstage(c,t1,t2) $\wedge$ speaking(c,t1,t2) $\supset$ main_character(c,t1,t2).
main_character(c1,t1,t2) $\wedge$ (c2 $\neq$ c1) $\supset$ should-pass-upstage-of(c2,c1).

and are written in Scene Constraint Language:

```
"is-maincharacter-over-interval "[c  t1  t2]
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>> determines if character c is the main character
>> over time interval [t1,t2]
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
IF          c  is-onstage-over-interval  [t1  t2]
```

```
AND         c  is-speaking-over-interval  [t1  t2]
THEN        c  is-maincharacter-over-interval  [t1  t2].
"should-pass-upstage-of "[c1  c2  t1  t2]
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>> determines if character c2 should pass upstage of
>> character c1 by the theater rule "don't pass
>> downstage of the main character in the scene"
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
IF          c1  is-maincharacter-over-interval  [t1  t2]
AND         c1 <> c2
THEN        c2  should-pass-upstage-of  c1.
```

So, the query:

```
Michael  should-pass-upstage-of  [y  [t1  t2]].
```

with facts:

```
LIST Characters Vito, Michael, Sonny.
FACT  Sonny  is-onstage-over-interval  [1  3].
FACT  Michael  is-onstage-over-interval  [2  8].
FACT  Vito  is-onstage-over-interval  [4  6].
FACT  Sonny  is-speaking-over-interval  [2  3].
```

will be resolved to:

```
Michael  should-pass-upstage-of  [Sonny  [2  3]].
```

Hierarchical Planning

Part of the general knowledge used to animate a scene in the Director's Apprentice is a "divide-and-conquer" planning strategy. This is based on *planning-space subdivision*,

whereby a large problem space (e.g. all of the actions that a character might perform) is divided into *loosely-coupled* subproblems, so that a planning decision affecting one will only minimally affect the others. (For example, in walking across a room, around a table and through a door, the subproblems "which side of the table to pass on" and "which hand to open the door with" are loosely coupled, while the subproblems "which door to exit through" and "what obstacles to avoid along the way" are closely coupled). The subproblems are organized into an abstraction hierarchy, so that the most general subproblems are solved before any of the details.

The hierarchical planner currently implemented in the General Knowledge Library generates a plan for each pair of key positions. In each plan, four major abstraction levels are defined:

Level	Goal	Constraints
First	Reach next position	Use key positions given in script
Second	Have Clear path	Avoid obstacles Stay on stage
Third	Satisfy Tendencies on Path	Approach attractive objects Avoid repellent objects Conform to rules of theater direction
Fourth	Animate gait pattern	Apply appropriate gait pattern for terrain Apply appropriate gait pattern for personality

A plan consists of a world model (general scene knowledge), the script, and a set of operator descriptions. One operator is *add-new-keypoint*, which inserts a new key position for a character between two existing key positions. This operator is activated when the straight-line path between two key positions can be improved, such as when an obstacle is in the path or when a significant tendency force is present to influence the character's movement. Another operator is *assign-gait*. This assigns gait patterns to the characters in the scene depending on such attributes as age, height and terrain.

Each rule in the Scene Constraint Language consists of a set of *antecedent* clauses followed by a number of *consequent* clauses. Each clause is either a function call (such as an expression evaluation), or an Object-Attribute-Value triple (such as "Vito is–grandfather–of Anthony").

The interpretation process is initiated by specifying a query that matches the consequent clause of some rule "r". When rule "r" is activated, the following actions take place:

1. If "r" is a function call, evaluate the function and return success;

2. Otherwise, rule "r" is (O_r A_r V_r):

a. for each antecedent clause "i"(O_i A_i V_i) in "r", (e.g. "i"=Michael son-of Vito), do:

(1) search for another rule "j" with consequent clause "c"(O_C A_C V_C) such that the Attribute of "c" matches the Attribute of "i", (e.g. "c"=*x* son-of *y*);

(2) bind $O_C \leftarrow O_i$ and $V_C \leftarrow V_i$, (e.g. $x_j \leftarrow$ "Michael", $y_j \leftarrow$ "Vito");

(3) with these bindings in place, recursively solve each antecedent clause in rule "j";

(4) if any clause in "j" fails (is unsolvable), fail rule "j":

(a) unbind the variables of "j", e.g. $x_j \leftarrow$ **NULL**, $y_j \leftarrow$ **NULL**;

(b) retract any partial conclusions recorded for rule "j";

(c) try again to solve clause "i" in rule "r", starting with the next rule after "j";

(5) otherwise ("j" succeeds), add clause "i" to the current context list of known truths;

b. if all antecedent clauses in rule "r" are solved, return success and record the consequent clause of "r" in the context list; otherwise, return failure.

Function-call clauses include display functions, memory management functions, list manipulation functions and the "assert" function for real-valued constraints. The "assert" function takes the form

(assert *clause update-strategy*)

where *clause* is an O-A-V triple, and *update-strategy* is one of "conserve", "revise" or "compromise". When an assertion is made, the current context list is searched for an already-accepted fact whose Object and Attribute match that of *clause*. If none are found, *clause* is added to the current context. If one *is* found, and its Value contradicts the Value of *clause* , then one of three actions are taken:

1. if *update-strategy* is "conserve", do nothing;

2. if *update-strategy* is "revise", retract the conflicting fact in the context list and replace it with *clause* ;

3. if *update-strategy* is "compromise", link *clause* to the "compromise list" associated with the conflicting fact. When this fact is later extracted, the compromise list will be processed and its net value (e.g. net tendency) returned.

Find-Path for Scene Animation

The robotics approach to finding a path between two locations in a room was discussed in Chapter 2 ([Lozano-Perez 83], [Brooks 83]). The main constraint on an autonomous robot is that a clear path must be found *automatically*. The Director's Apprentice is intended to reduce animation effort by 80-90%, not 100%, so instead of a method that guarantees success in all situations, a fast, simple algorithm that works in an uncluttered stage environment is employed. The method is illustrated in the following example.

Character "c" must reach point X past an obstacle (Figure 4-5 through 4-10).

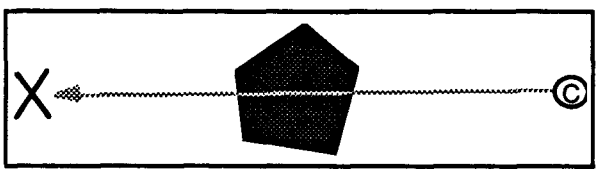

Figure 4-5 Character C approaches X.

Step 1: Construct a bounding box around the obstacle, aligned along the path.

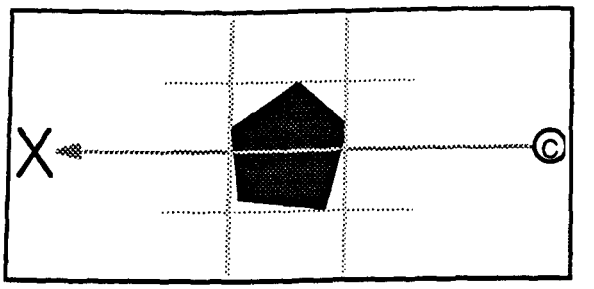

Figure 4-6 Bounding Box

Step 2: Locate four possible key positions on the outside corners of the bounding box. This defines two possible clear paths around the object, (C-C_{11}-C_{12}-X) and (C-C_{21}-C_{22}-X).

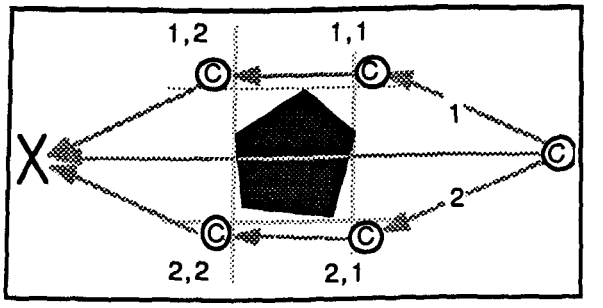

Figure 4-7 Two possible routes.

Step 3: Determine the net tendency vector for each new key position, using the appropriate displacement functions.

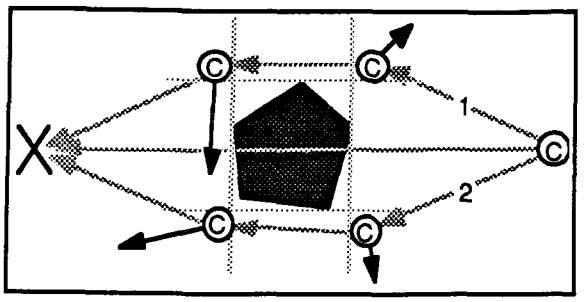

Figure 4-8 Tendencies for Each Position.

Step 4: Form the vector sum of the tendency vectors to determine the net tendency with respect to the perimeter of the obstacle.

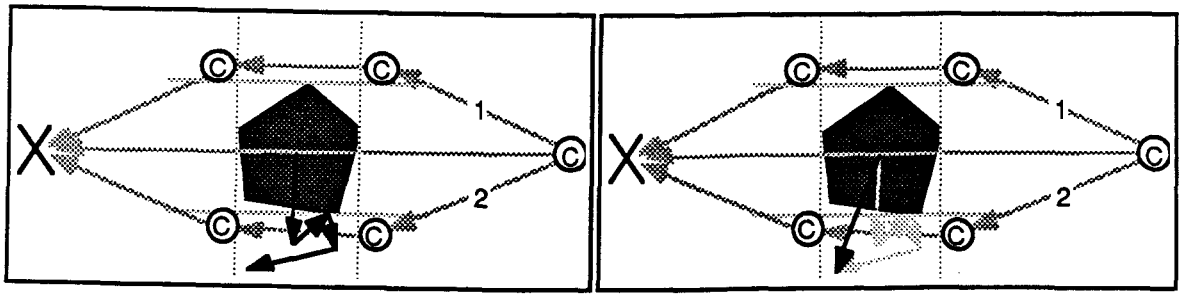

Figure 4-9 Determining Net Tendency

Step 5: Select path on the basis of the direction of the net tendency. In this case, the net tendency vector pointed towards the path 2. The tendency vector for each key position displaces the path around the obstacle. The grey line is the final solution.

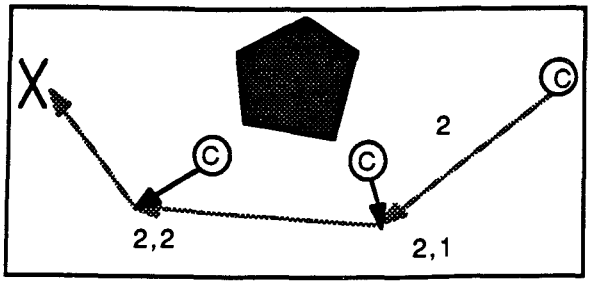

Figure 4-10 Choose Path

Expressed in scene constraint logic[1]:

```
"has-path[c  x]"
{
        [c x] forms-path *p.
        ∃o∈ objects
            (p intersects o) ⊃
            {
                [p, o] has-bounding-box [*top,*right,*bottom,*left].
                [top,right,bottom,left]  form_candidates[*c11,*c12,*c21,*c22].
                ∀k∈ [c11,c12,c21,c22] { k has-net-tendency *tk}.
                (vector-sum([tc11, tc12,tc21,tc22] has-same-side-as [c11,p]) ⊃
                    {[c x] has-path [c c11 c12 x]}
                        ; {[c x] has-path [c c21 c22 x]}.
            } ; [c x] has-path [c x].
}
```

This is a simplified outline of the algorithm. An outer shell must be provided to recursively check all resulting paths for new obstructions, and there are some situations that this approach will not handle without extensions. These include:

(1) A straight-line path with strong tendencies, but no obstacles (Figure 4-11).

---

1 Here an asterisk indicates a result variable, and the semicolon represents an "else" choice. Scene Constraint Language substitutes "exists" for "∃".

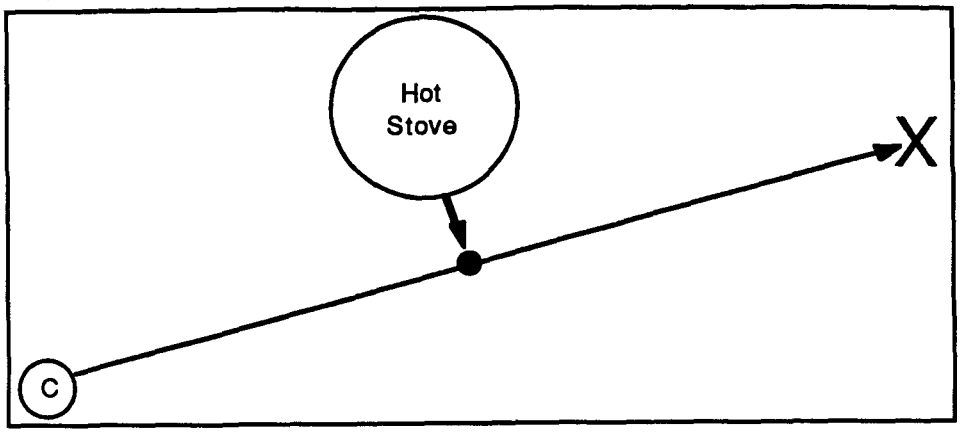

Figure 4-11 Tendencies without an obstacle

Here the character has a strong tendency to avoid the hot stove, even though it is not actually an obstacle.

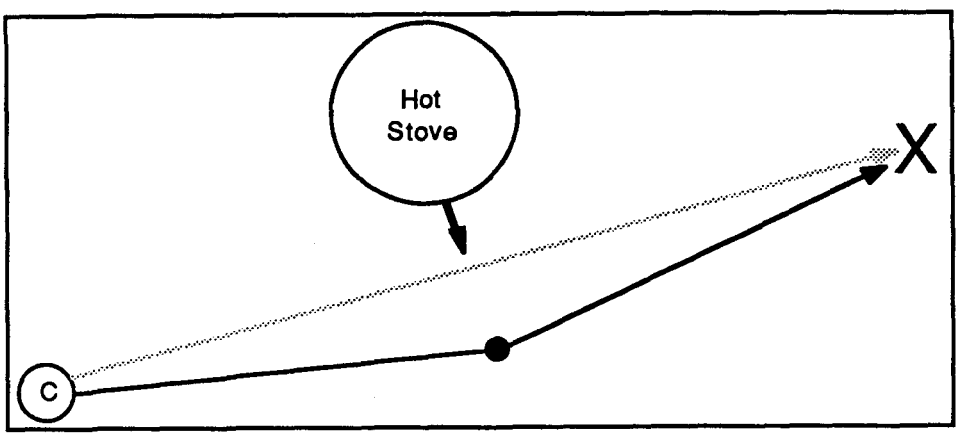

Figure 4-12 Path subdivided and bent by tendency

This can be handled by subdividing the path, and deviating the midpoint according to its net tendency. In general, all paths (including ones generated by higher-level steps) should be subdivided this way, until some minimum threshold distance for each step is reached.

(2) Two repellent forces on either side, causing a net force vector which does not lie to either side of the original path. (Figure 4-13).

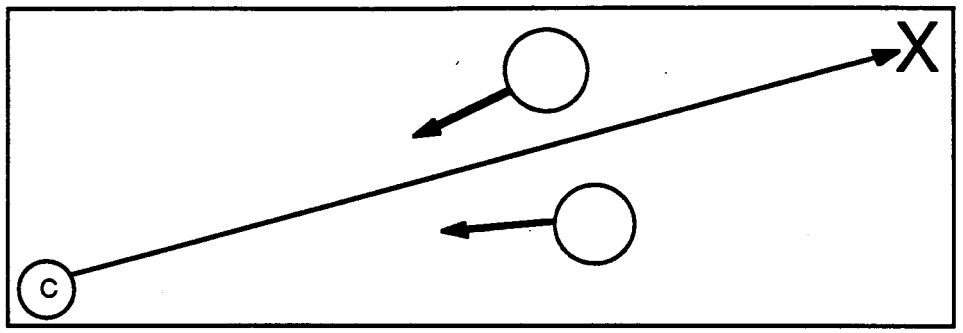

Figure 4-13 Balanced tendencies

This can be handled by enlarging the repellent objects to an area that includes their significant force areas. The total area now constitutes an obstacle.

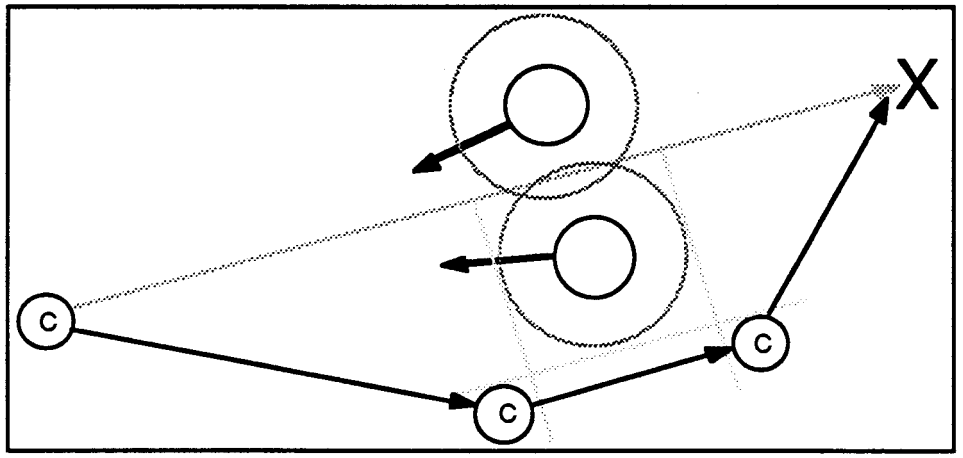

Figure 4-14 Balanced forces resolved to an obstacle

(3) An obstacle with no significant tendencies. In this case, a tendency to take the shorter path is generated by generating two tendency vectors proportional to the perpendicular distance from the path to the edge of the obstacle.

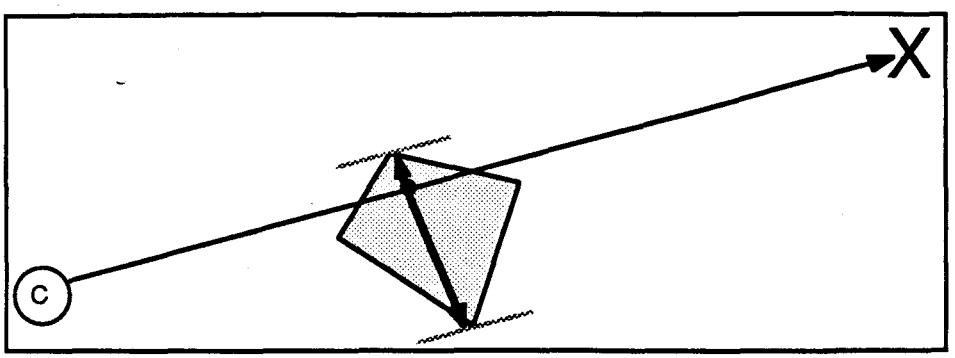

Figure 4-15 Tendencies from distance

When inverted, a tendency to take the shorter path is produced.

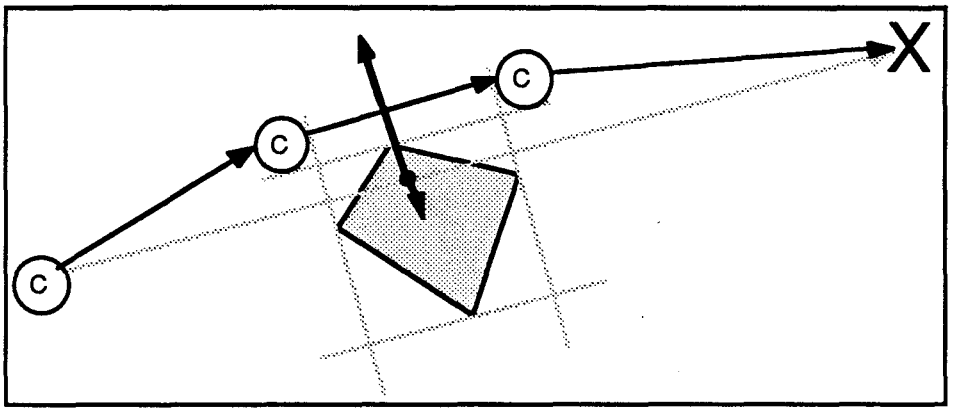

Figure 4-16 Shorter distance prevails

Planning Example

To illustrate the use of the planning routines in the General Knowledge Library, suppose the following script were to be animated.

Script Fact Collection (includes):
Characters: Tex(T), Black Bart(B) and Miss Kitty(K).
Setting: as in Figure 4-17.

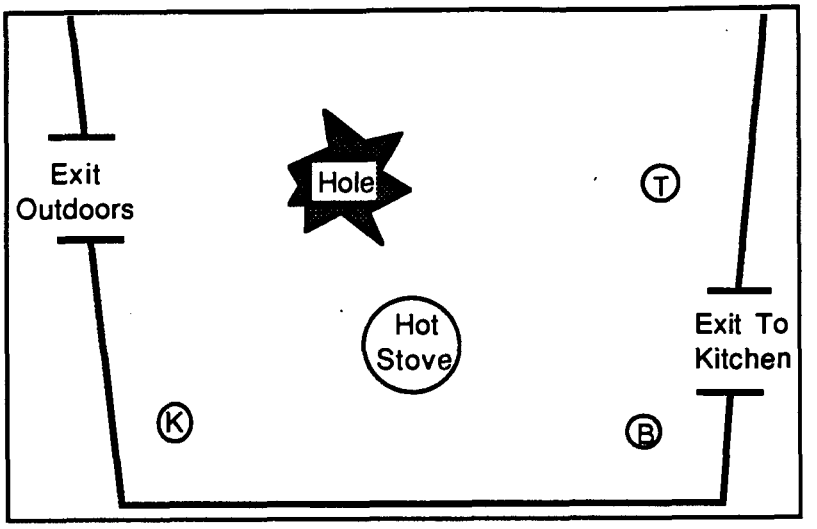

Figure 4-17 Setting

Relationships:

(1) Tex likes Miss Kitty.
(2) Black Bart desires Miss Kitty.
(3) Miss Kitty desires Tex.
(4) Miss Kitty is afraid of Black Bart.
(5) Black Bart hates Tex.
(6) Tex hates Black Bart.

Action (director's annotations in italics):

(1) Tex says,"Bart, you dirty pole-cat. You'd better be goin'."
(2) Black Bart grunts and exits outside.
Black Bart is bowlegged. Black Bart is angry.
(3) Miss Kitty says "Going to stay awhile, Tex?"
(4) Tex says "No, ma'am. Got some dogies to corral. See ya'll."
(5) Tex exits outside.
Tex is shy.
(6) Miss Kitty sighs and exits to the kitchen.
Miss Kitty is sad.

General Knowledge Library (includes):

(1) Hot stoves are obstacles.
(2) Holes in the floor are obstacles.
(3) Characters tend to avoid hot stoves.
(4) Characters tend to avoid holes in the floor.
(5) People who are angry tend to use gait "stamp".

(6) People who are shy tend to use gait "walk-quickly".
(7) People who are sad tend to use gait "shuffle".

Each line of the script represents a snapshot of the action in the scene, and translates directly into a set of key positions for the characters, together with suggestions for the constraints on their actions to the next position (next line).

Figure 4-18 depicts the plan of Black Bart exiting.

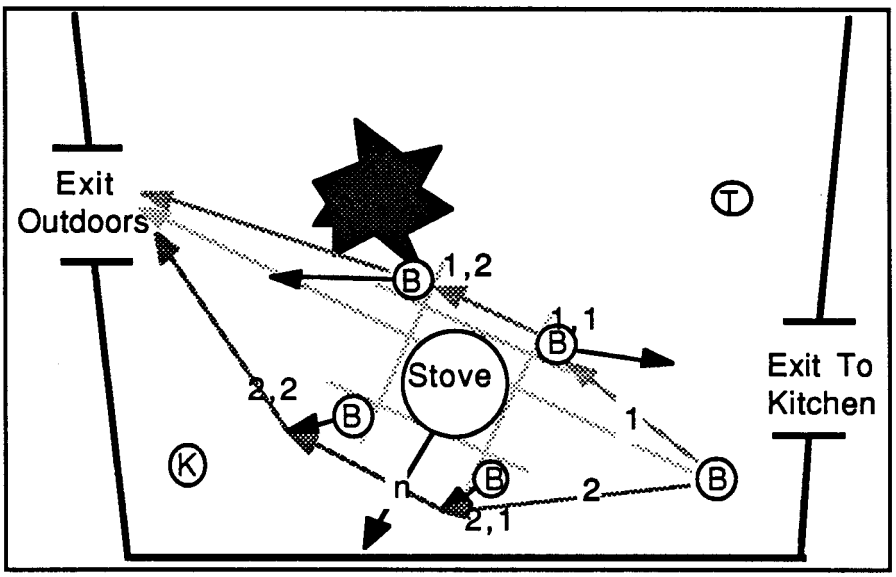

Figure 4-18 Black Bart exits

General Knowledge library statements (1) to (4) are used to activate operator *insert-keypoint* to go around the obstacles Each of the new candidate keypoints has a tendency pointing away from the stove and the hole, since they are obstacles that should be given a

wide berth. The facts that Bart desires Miss Kitty and hates Tex also affect the tendencies from each position. Since the sum-vector for the four key positions ("n") points up-stage-left, the upstage path (path 2) around the tendency displacements away from Tex and towards Miss Kitty is chosen. The gait pattern for Black Bart's walk is chosen as a compromise between "bowlegged-gait" and "stamping-gait".

Figure 4-19 depicts Tex exiting. Since Black Bart is gone, only the repulsion of the stove and the attraction to Miss Kitty affect the tendency vectors as Tex goes around the hole. Since the stove is much closer to Tex than Miss Kitty is (and his attraction for her is weak anyway), she does not affect the net tendency downstage (path 1). The director's annotation "Tex is shy" along with general knowledge entry (6) is used to assign a quick gait to Tex.

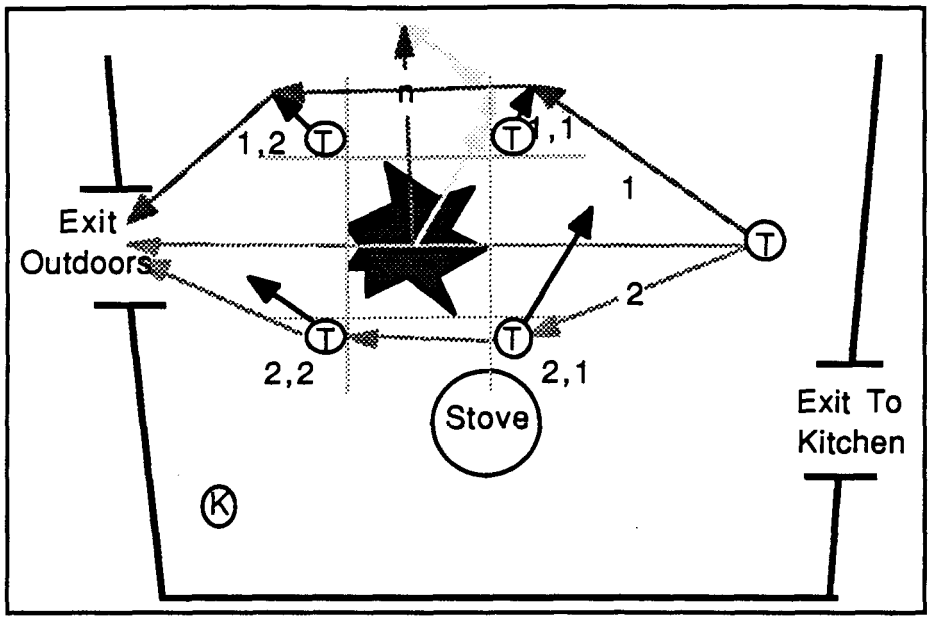

Figure 4-19 Tex exits

Figure 4-20 shows the route planned for Miss Kitty. Since neither Tex nor Black Bart are in the scene, her path is influenced only by taking the shorter of the two paths (path 2). Since the director annotated the fact that she was sad in this scene, the General Knowledge Library statement:

(7) People who are sad tend to use gait "shuffle".

is used to assign "shuffle-gait" to her walk.

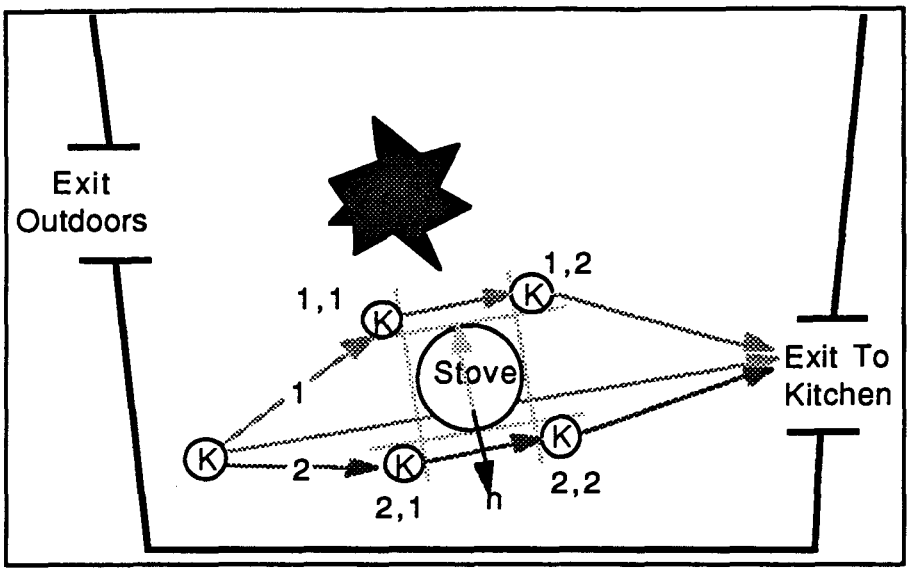

Figure 4-20 Miss Kitty exits

Path Animation

The actual animation of human characters walking, running , climbing and descending stairs, and so on, is achieved by associating with each planned path a *gait sequence* derived from statements in the the Script Fact Collection (one character may run, another may limp) and from statements in the General Knowledge Library (people generally *hop* up each step in climbing stairs, but *drop* down each step when descending). A gait sequence is composed of standard *gait patterns* taken from the GKL, each one consisting of a set of functions defining the motion of some part of the body through space relative to a frame of reference. The walking gait, for example, has three functions: an up-and-down

pattern for the center of mass (relative to the floor), a forward motion of the unweighted foot (relative to the weighted foot) and a pattern of angles for the unweighted ankle as it moves through space. Figure 4-21 shows these three patterns.

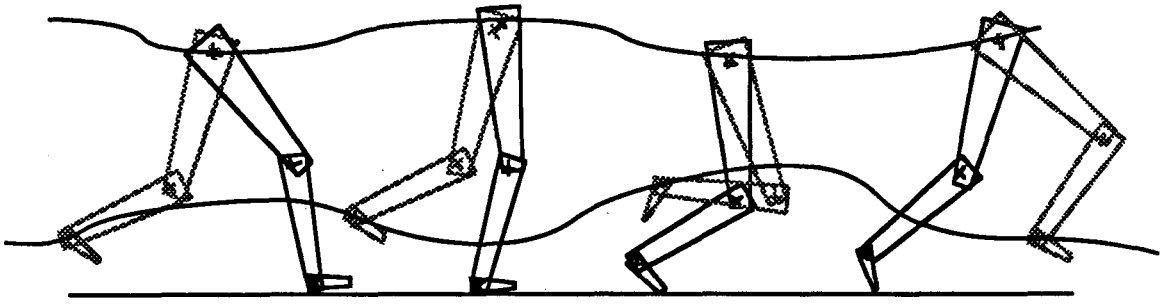

Figure 4-21 Gait pattern data for walking

All other angles and positions (the knee-joint angles, the hip-joint angles, the waist location) are determined by propagating the gait pattern data and the known constraints on bodies. This data drives an inverse kinematic walking program that displays the characters in real time on the screen of an IRIS 2400 workstation. The gait patterns are inverse kinematic motion descriptions of the path since the Cartesian coordinates of the joint positions are known, and joint angles are calculated from trigonometric relationships. Since simplifying assumptions based on human anatomy are used (e.g., the knee joint has only one degree of freedom, and only a limited range of movement) this calculation proceeds in real time. Note that the general inverse kinematic methods of Girard ([Girard 86]) permit more complex body models at the cost of slower

processing.

Other character movements (such as arm gestures), can be specified similarly, although these are not part of the current implementation.

Implementation Details

The Scene Constraint Interpreter is currently implemented in "C" on a SUN workstation. A graphical interface allows interaction via pop-up menus that choose the script file and the goal, and to interact with various debugging features. The Script Fact Collection and General Knowledge Library are written as Script Constraint Language statements, and entered using an ordinary text editor. The result of a scene analysis is a disk file that, for each character, gives a sequence of path segments and a gait pattern for each segment. This file is transferred via Ethernet to the IRIS 2400 which displays the characters in action on the stage set.

A number of problems and limitations exist in the current implementation. Some of these are listed below with suggestions for improvements.

1. The user interface, described above, is too hard for non-"computer literate" directors and animators to use.

   Possible solutions:

   (a) Incorporate a structured rule editor for constraint writing, having an icon-driven interface that allows the user to build up constraint rules interactively. The Scene Constraint Language now effectively becomes a "machine-language" for implementing scene constraints, much as high-level expert system shells such as OPS([Forgy 77]) shield users from LISP.

   (b) Improve the debugging facilities. Currently, stepping through the rule firings and interrogating the values of plan variables is all that is available.

2. The General Knowledge Library contains no constraints to handle the following cases:

   (a) Two key positions are separated by an obstacle that is close to the edge of the stage, and the net tendency would lead the character over the edge.

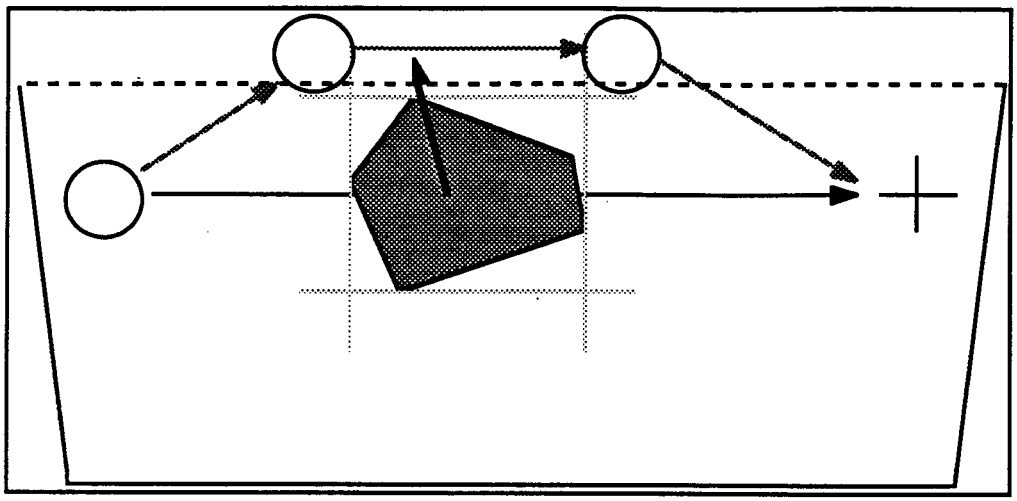

Figure 4-22 Edge-of stage failure

Instead of choosing a second-choice route, the tendency constraint will return failure.

Possible solution: Assign strong repulsion tendencies to the edge of the stage, so that the other route around the obstacle will be preferred. Alternately, include a special rule that prevents the generation of off-stage candidate key-points.

(b) An obstacle has two other obstacles on either side of the path, so that neither path can be taken. System returns failure.

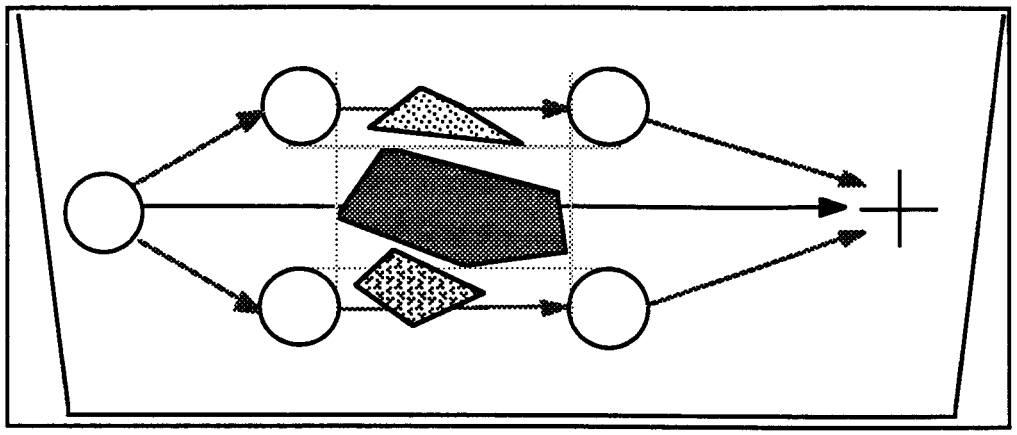

Figure 4-23 Closely-spaced obstacles

Possible solution:

Coalesce obstacles that are closer to each other than the diameter of the character into one large obstacle.

(c) If two character paths intersect, there is no direct control over whether they will meet or miss. This is the "rendezvous problem".

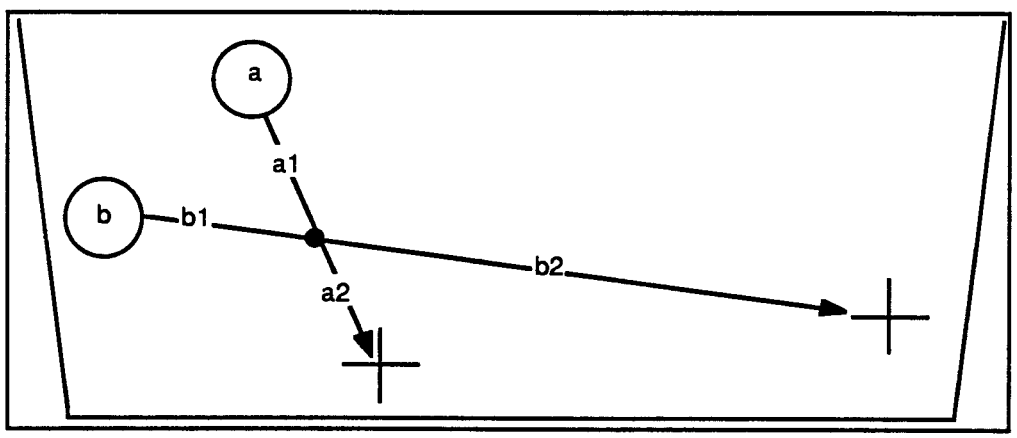

Figure 4-24 Rendezvous Problem

For example, in Figure 4-24, characters "a" and "b" must meet at the intersection of their paths, that is, the transit time for segment "a1" must be the same as segment "b1" (assuming that they start at the same time).

Possible solution:

Associate with each gait pattern a *tendency* (probability density function) to move at a certain rate.

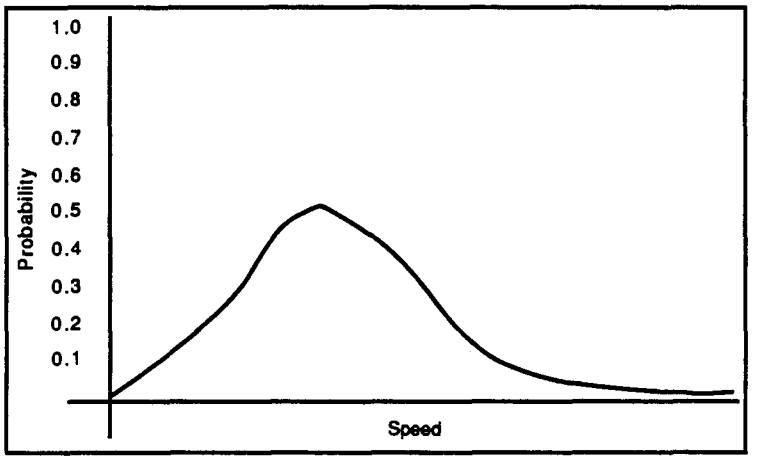

Figure 4-25 Tendency function for rate

Initially, assign to each gait pattern segment the expected value for the rate of locomotion. Then, determine if the first meet/miss goal is attained, and if not, pick the 90% peak probability rate for one character (some should be made faster than the expected value, some slower). If that does not work, try the 90% peak probability rate for another character, and so on. Each step alters the rate of some character, but only in small steps away from the most probable (most reasonable) rate for that character. Eventually the first

meet/miss goal will be achieved (if at all possible), and the next goal can be solved in the same manner.

3. A bounding box is too crude for some odd-shaped objects, causing a failure to find a path in some cases.

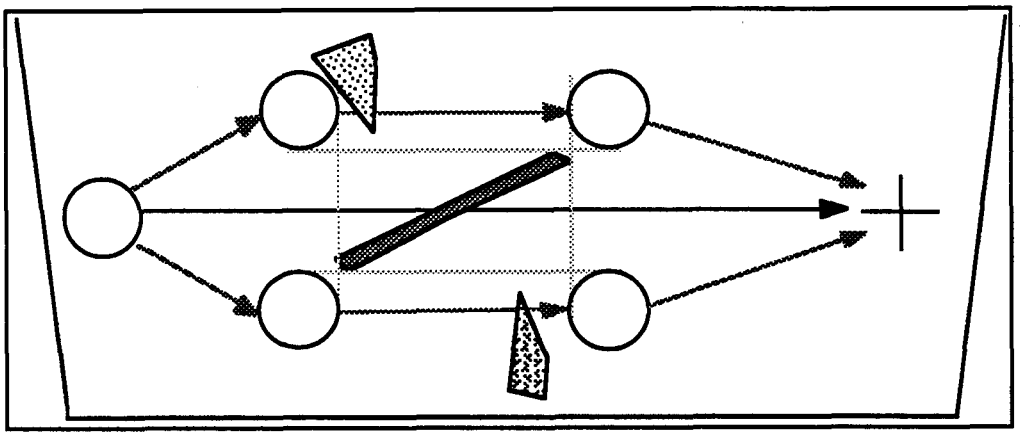

Figure 4-26 Bounding box with poor fit to object

Possible Solution:

Instead of a bounding box, the convex hull of the object may be used, and a new keyposition associated with each vertex. This is much more computationally expensive.

4. The figures are assumed to be circular when seen from above, and are of a fixed diameter for each character. While this does allow for different-diameter characters, it does not allow for variations in the effective diameter, such as when the arms are extended at the sides.

Possible Solution:

Allow for a different diameter at each keyframe, which may be interpolated. Even more flexible would be the use of ellipses to represent the characters, whose principle axis could be oriented by the animator at each keyframe.

5. The rendering of the animated figures, while fast, is crude (all figures are drawn in wire frame, and have no head or arms).

Possible Solution:

Use solid shading with spherical joints to produce more life-like renderings of the figures.

6. The action, although physically reasonable, may be aesthetically lacking.

Possible Solution:

Many differences between aesthetically "good" action and aesthetically "poor"

action derive from how well the scene adheres to the established principles of theater direction, as described in Chapter 1. By encoding some of these principles in Scene Constraint Language, and using them to generate tendencies for the figures, better quality action may result. For example, if the system derived the fact that a character is the main character in the scene, tendencies could be established to draw him or her to a "strong" stage position, or to move in a "strong" way (see [Brown 36] or [Allensworth 82]).

Summary

Three main points may be concluded from my experience in implementing the Director's Apprentice:

1. Although the Director's Apprentice is not a completely implemented expert system at present, it conforms to one of the most critical aspects of an expert system: the strict separation of procedural and problem-specific knowledge. Lacking is a sophisticated user interface that would allow non computer-literate animators access to the rule bases.

2. While most logic-based systems cannot deal directly with conflicting continuous-valued constraints, the Director's Apprentice uses tendency compromise to implement a path finding algorithm that is fast and character-specific.

3. The current implementation, while incomplete, may be expandable into a useful scene animation tool by incorporating the improvements described in the previous section.

Chapter 5:
Conclusions

This chapter discusses some conclusions resulting from the research in this thesis, particularly its implications for animators, directors, and for graphics and AI researchers. Also discussed are some problems with the current approach and some future extensions.

Four original goals of this thesis were described in Chapter 1. The first goal was to investigate some kinds of scene-level constraints that may be useful in implementing a figure animation system. The conclusion is that scene-level constraints are most important for relieving the animator from concern about unlikely or impossible actions, such as following an unreasonable path, that may occur between keyframes. Even simple constraints, preventing the path of a character from intersecting with other objects in the scene, are an important addition to a figure animation system.

The second goal was to investigate extensions to logic programming for dealing with quantitative reasoning. This is provided in the Scene Constraint Language by a consistent mechanism to relax conflicting continuous-valued assertions. This mechanism has been incorporated into a resolution theorem prover to provide both discrete (logical) and

continuous (quantitative) reasoning in one system.

The third goal was to investigate the application of an extended logic programming tool for defining a hierarchy of constraints on animated figures. This was achieved in the Director's Apprentice by defining a series of goals, each one representing one level of the planning hierarchy. These goals are satisfied by using a combination of discrete and continuous reasoning.

The fourth goal was to assess if the prototype figure animation system developed in the Director's Apprentice project has qualitative advantages over conventional animation systems. One qualitative advantage displayed by the prototype system is the ability to automatically alter the action of the figures in the scene to suit changing settings, even after the figure animation has been specified. This implies a new type of flexibility in "what-if" style testing for different settings and scenes.

The implications for potential users of the system are several. Time pressure is a part of every animator's job. An animation system that will save time by automatically taking care of character movement constrained by physical limitations and good scene direction — regardless of revisions to the setting or the action — would be an important asset. An interface must also be provided that will allow the animator to develop the artistic aspects of the scene, while shielding her from logic programming and gait pattern design.

Theater directors and set designers could also make good use of a system that can generate animated "story-boards" of a script, maintaining reasonable action as set designs are revised. Since computer graphics easily provides multiple viewpoints (including that of the director, the audience, each actor, and so on) and, with appropriate rendering software, different lighting and colours, the ability to try out many different scenes quickly and with minimal knowledge of animation could provide a significant design tool for the theater.

To use the system, the animator (or director, designer) starts by describing the category of the setting (type of room or area). A character list is added, including details of personality types that might affect the interpretation. Next, actions in the script are added, describing character movements across the stage together with any gestures to be displayed. Character positions are established using drag-to-place on the graphics screen. Annotations, describing goals, inter-character feelings, and tendencies are added next, followed by any setting details needed to form a crude story-board. These story-boards become keyframes for the system to interpolate. With preparations complete, the scene interpreter is executed, producing a 2-D plan view of one interpretation of the scene, using script annotations and General Knowledge Library constraints to produce only reasonable actions. The action plan displays the character paths and tendency vectors as illustrated in Figure 5-1 (from 4-15).

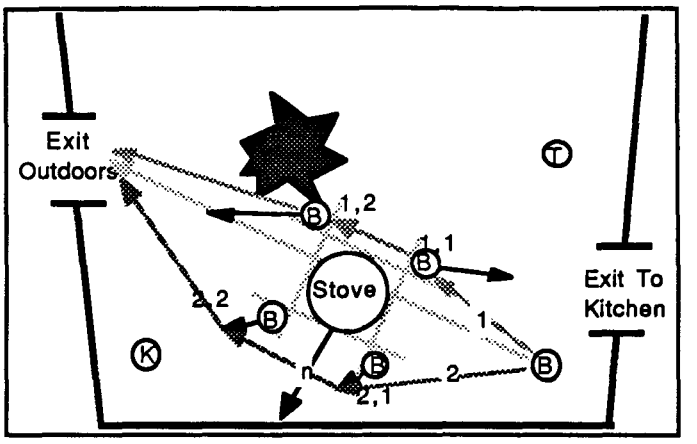

Figure 5-1 Display of Action Plan

At this point, the animator may disagree with the interpretation and will decide to revise the action by adding more key positions. However, this will not improve the competence of the scene interpreter — which is the long-term goal — so instead, the user may trace the flow of decisions made by the logic interpreter to determine why the scene was generated the way it was. Once the cause of the undesired action has been determined, script annotations and tendency constraints may be edited to provide a revised interpretation. These edited characteristics may then be archived for later reference.

The path that a character follows in performing an action from the script determines much about more detailed actions, such as the gestures that he makes as he walks. For example, two characters may shake hands or fight only if their paths happen to cross. The prototype Director's Apprentice has addressed only path problems so far since they are at the highest level in the planning hierarchy and must be dealt with first . Planning rules for lower-level actions in the hierarchy, such as gestures, will be addressed later.

The rule base currently implemented contains approximately thirty rules of, on average, five clauses each. These rules implement the planning constraints of the "Tex and Miss Kitty" scene of Chapter 4. With them, several different scenes have been generated by altering the locations of the characters, the locations of the props, and the relationships between the characters. Characters who are speaking do not have other characters pass in front of them, but apart from that, no theater direction rules have yet been implemented. The number of rules in a knowledge base is limited only by processing time. Currently, each new scene requires about ten minutes of processing on a SUN-3 workstation, but is expected to be faster in an improved version of the logic interpreter. Experiments involving really large rule bases (hundreds of rules) will establish the practical limitations of the system.

Once the plan view of the action is accepted, gait patterns based on scene constraints (such as terrain or rendezvous requirements) and on script annotations (sad walks, nervous walks) are generated by the interpreter and transferred to the rendering system, where they can be viewed in wire-frame at real time speeds. A solid rendering may also be generated if it is needed.

In computer graphics research, major progress has been made in the past decade toward perfecting the rendering of still scenes; further progress being largely dependant on improvements in hardware performance. The implication of this thesis for computer

graphics research is that computerized figure animation — only barely practical today — may emerge as a new area of research and development if the laborious nature of action specification can be reduced. Existing specification methods — working at a 'machine language' level — are only practical when a small number of parameters (five or six) change in each frame. Hundreds of parameters are required to animate a group of people who interact in a scene. With high-level action specifications, made possible through constraint satisfaction techniques, the generation of complex animated scenes involving human and animal characters may become common.

AI research into knowledge-based systems has, for the most part, treated continuous-valued (quantitative) reasoning separately from inference (discrete reasoning), partly because exhaustive search (commonly used in discrete reasoning), is impractical with quantitative problems. An approach to integrating continuous and discrete reasoning in figure animation was explored in this thesis, and may have other applications. For example, knowledge-based systems are being increasingly applied to problems in process control; these also involve continuous measurements (of temperature, flow, power, and so on), that must be reasoned with to determine optimum control strategies.

This thesis has also taken the AI field of knowledge engineering into a new area. One of the key problems in any knowledge engineering situation is the acquisition of knowledge. Currently, all general scene knowledge in the Director's Apprentice derives either from

textbooks on theater direction, or from common sense. However, future improvements to the competence of the interpreter will have to come from human experts (animators or theater directors) who can say "This looks wrong, because character '*x*' would not behave like that in this scene. Instead, he would probably do this ... ". A knowledge engineer, conversant with logic programming, would then formulate new constraints (and edit old ones) to satisfy and generalize the new advice.

A number of extensions and improvement will be found in future versions of the Director's Apprentice. One improvement would be to provide meta-logical functions in the Scene Constraint Language for *property inheritance*, allowing assertions to be made about whole classes of attributes, which may subsume other classes, and so on. For example, the following lines illustrate how the system could reason about a character classes:

class villains **has-property** hates(heroes).
Black-Bart ∈ bad-guys.
Tex ∈ good-guys.
bad-guys ⊃ villains.
good-guys ⊃ heroes.

⇒ Black-Bart hates(heroes).

⇒ Black-Bart hates(Tex).

Currently, gait patterns are designed manually, and entered as coordinate triples. Another useful extension, then, would be a graphical tool for designing and naming new

patterns, to be incorporated into a gait pattern library. A facility for specifying upper-body gestures would also be useful, possibly employing a reach-hierarchy mechanism for grasping tasks.

One important part of the General Knowledge Library is the planning hierarchy which guides the problem-solving tasks in selecting appropriate paths for the characters. The planning hierarchy is currently specified by a sequence of logic (Scene Constraint Language) clauses. Sacerdoti employed an explicit plan hierarchy([Sacerdoti 77]) called a "procedural net" for solving blocks-world assembly problems. A similar planning structure, developed with the aid of a graphical editor, might make plan design and visualization easier for animation problems.

One definite improvement that should be made to subsequent versions of the Director's Apprentice is to re-implement it in another high-level language. 'C' was chosen for the current version since it is the standard language of graphics systems and is portable across most graphic workstations. However, the difficulties involved in implementing complex, transformable data structures in 'C', and the lack of automatic storage management, made the inference engine very complex to develop and difficult to extend. It is expected that subsequent versions will likely be written in LISP. LISP has excellent structuring facilities, while still providing reasonable portability and greater flexibility than pure-logic systems like PROLOG.

In the future, the Director's Apprentice is envisioned as a complete expert system, containing a user interface tailored to the needs of animation directors. Instead of re-formulating rules in Scene Constraint Language every time a new situation arises, the user may define scenes and characters by choosing from a set of "stock types". These stock characters, sets and scenes will be modified interactively until the desired effect is achieved. More ambitious users may want to alter the knowledge base directly to see the effect of adding new constraints to the planning hierarchy. Very complex actions, such as dancing or sword-fighting may not be generated fully automatically, but rather will be treated as details that the animator can touch up by hand once the general flow of the scene is established.

In summary, the Director's Apprentice project has demonstrated the need for constraint-based mechanisms for specifying human animation. In opinion of the author, no specification method can be truly useful for animating characters in a scene unless the environment of the characters is accounted for automatically. A logic-language based planning system, having extensions for quantitative reasoning, appears to be a promising approach.

Appendix:
Grammar of the Scene Constraint Language

A set of constraints in the Scene Constraint language is called a rule base. The BNF grammar of a rule base is as follows. Each grammatical unit is separated by a non-zero length whitespace string. Comment lines begin with '>>'.

```
<rule base> ::=
      <assertion-unit><assertion-unit>*

<assertion-unit>  ::=
    <list-definition>|
    <rule-definition>

<list-definition>::=
    'LIST'  <list-name>  <list-element><list-element>*  '.'

<list-element>::=
    <string-token>|
    '('  <list-element>*  ')'

<rule-definition>::=
    <rule-name>  <variable-declaration-list><rule-body>'.'

    <rule-name>::=
        '"'  <string-token>  '"'

    <variable-declaration-list>::=
        <empty>|
        '['  <variable-name><variable-name>*  ']'

    <rule-body>::=
        <antecedant-list> 'THEN' <consequent-list>

    <antecedant-list>::=
        <if-clause>  <and-clause>*

    <if-clause>::=
        'IF' <clause>
```

```
<and-clause>::=
    'AND' <clause>

<clause>::=
    <goal-clause>|
    <function-clause>

<goal-clause>::=
    <attribute > <predicate> <value>

<function-clause>::=
    '(' function-name <arglist> ')' |
    { expression relop expression }

<expression>::=
    arithmetic expression |
    string expression

<relop>::=
    '=' | '<>' | '<' | '>'

<function-name>::=
    <string-token>

<arglist>::=
    <function-arg>*

<function-arg>::=
    variable-name |
    is-goal

<is-goal>::=
    <attribute>

<attribute > ::=
    <string-token> | <list> | <list reference>

<predicate>::=
    <string-token>

<value>::=
    <string-token> | <list> | <list reference>

<list> ::=
    '(' list-element* ')'

<list reference> ::=
    <string-token><sublist-identifier><sublist-identifier>*

<sublist-identifier>::=
    '.' <integer>
```

Example

The following fragment from a Scene Constraint Language text illustrates all of these constructs.

```
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>> List describing each character, and for each, its starting position in
>> key-frame 1 and its goal position in key-frame 2.
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
LIST character-list  (Tex (3.0 2.4) (3.6 4.1))  (MissKitty (2.1 4.2)(6.1 5.0))
(BlackBart (3.2 5.5) (7.1 6.0)).

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>> List describing each stage prop, and for each, its location and
>> size.
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
LIST props-list  (couch (5.5 7.1) (4.0 2.0))  (table (1.0 2.0) (2.5 2.5))
(chair (3.1 2.2) (0.8  0.8)).

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>> Rule that, for each character, subdivides the path by trying to determine
>> if a clear path from start to goal exists. If so, an assertion to that effect
>> is added to the current context.
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
"subdivide path"[c]
IF   (exists character-list "c")
>> "exists" is a function that generates each element of the list in argument 1,
>>   assigning it to argument 2.  The loop is terminated by "end-exists".
AND        (c.1 c.2) has-no-obstacle-on props-list
AND        (assert c has-clear-path (c.1 c.2))
AND (end-exists)
THEN path-subdivided is true.
```

References and Bibliography

[Allen 83]

James F. Allen, "Maintaining Knowledge about Temporal Intervals", CACM 26 (11), 1983, pp. 823-843

[Allensworth 82]

Carl Allensworth, *The Complete Play Production Handbook (Rev. Ed.)*, Harper and Row, Inc., 1982.

[Armstrong 86]

W.W. Armstrong, M. Green and R.Lake, "Near-Real-Time Control of Human Figure Models", in Proceedings of Graphics Interface Conference, 1986, pp. 147-151

[Badler 78]

N. Badler, R. Bajcsy, "Three-Dimensional Representations for Computer Graphics and Computer Vision", Computer Graphics, Vol 12, 1978, pp. 153-160

[Badler 79]

N. Badler and S. Smoliar, "Digital representations of human movement", ACM Computing Surveys, Vol. 11, pp. 19-38

[Badler 80]

N. Badler, J. O'Rourke and B. Kaufman, "Special problems in human movement simulation", Computer Graphics, Vol 14, pp. 189-197

[Badler 82]

N. Badler, "Modelling the Human Body for Animation", IEEE Computer Graphics and Applications, (Special Issue), Vol.2, November 1982

[Badler 85]

N.I. Badler, J.D. Korein, J.U. Korein, G.M. Radack, L.S. Brotman, "Positioning and animation human figures in a task-oriented environmment", The Visual Computer, Vol. 1 (1), Springer-Verlag, 1985, pp. 212-220

[Badler 86]

N. Badler, "Animating Human Figures: Perspectives and Directions", in Proceedings Graphics Interface Conference, 1986, pp. 115-120

[Badler 87]

N. Badler, K. Manoocherhri, G. Walters, "Articulated Figure Positioning by Multiple Constraints", IEEE Computer Graphics and Applications, (Special Issue), Vol.7(6), June 1987

[Barr 80]

A. Barr and J. Davidson, "Representation of Knowledge", Memo HPP-80-3, Stanford University, 1980

[Barr 81]

A. Barr, P. Cohen and E. Feigenbaum (eds.), *The Handbook of Artificial Intelligence*, William Kaufman Inc., 1981

[Benedetti 85]

R. Benedetti, *The Director at Work*, Prentice-Hall, 1985

[Borgida 84]

A Borgida and T. Imielinski, "Decision Making In Commitees", in Proceedings of AAAI Conference on Non-Monotonic Reasoning, New Paltz NY, October 1984

[Borning 79]

Alan Borning, "ThingLab:A Constraint-Oriented Simulation Laboratory", Xerox PARC Report SSL-79-3, Xerox Corporation, 1979

[Brachman 85]

R. J. Brachman, "I Lied about the Trees, Or, Defaults and Definitions in Knowledge Representation", AI Magazine, Fall 1985

[Brooks 83]

Rodney A. Brooks, "Solving the Find-Path Problem by Good Representation of Free Space", IEEE Transactions on Systems, Man and Cybernetics, Vol SMC-13, No. 3, March/April 1983, pp. 190-196

[Brown 36]

G. Brown and A. Garwood, *General Principles of Play Direction*, Samuel French Ltd., 1936

[Buchanan 84]

B.G. Buchanan and E.H. Shortliffe, *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, Addison-Wesly, 1984

[Burtnyk 76]

N. Burtnyk and M. Wein, "Interactive skeleton tehcniques for enhancing motion dynamics in key frame animation", CACM Vol. 19 (10) (Oct 1976), oo. 564-569

[Calvert 80]

T. Calvert, J. Chapman, A. Patla, "The integration of subjective and objective data in the animation of human movement", Computer Graphics, Vol. 14, 1980, pp.198-203

[Calvert 82]

T. Calvert, J. Chapman, A. Patla, "Aspects of the kinematic simulation of human movement", IEEE Computer Graphics and Applications, Vol. 2, pp.41-50

[Calvert 83]

T.W. Calvert, "Computer assisted filmmaking: A review", In Proceedings of Graphics Interface Conference, 1983, 263-270

[Clocksin 81]

W. Clocksin, C. Mellish, *Programming in Prolog*, Springer-Verlag 1981

[Cohen 85]

J. Cohen, "Describing PROLOG by its Interpretation and Compilation", CACM (28), December 1985

[Davis 82]

R. Davis and D. Lenat, *Knowledge-Based Systems in Artificial Intelligence*, McGraw-Hill 1982

[Davis 82b]

R. Davis, "Expert Systems: Where are We? And Where Do We Go From Here?", AI Magazine (2), 1982

[Dean 85]

Thomas Dean, "Temporal Reasoning Involving Counterfactuals and Disjunctions", in Proceedings of IJCAI-85, 1985, pp.1060-1062

[Dechter 85]

Rina Dechter and Judea Pearl, "The Anatomy of Easy Problems: A Constraint-Satisfaction Formulation", in Proceedings of IJCAI-85, 1985, pp.1066-1072

[Delgrande 86]

James P. Delgrande and John Mylopoulos, "Knowledge Representation: Features of Knowledge", in *Fundamentals of Man-Machine Communication: Speech, Vision and Natural Language*, Cambridge University Press, 1986.

[Dooley 82]

M. Dooley, "Anthropometric modeling Programs- A Survey", IEEE Computer Graphics and Applications, Vol. 2, pp. 17-25

[Doyle 79]

J. Doyle, "A Truth Maintenance System", Artifical Intelligence , Vol12, 1979, pp. 231-272

[Drewery 86]

K. Drewery and J. Tsotsos, " Goal Directed Animation using English Motion Verbs", in Proceedings of Graphics Interface-86, 1986

[Duda 84]

R. Duda and R.Reboh, "AI and Decision Making:The PROSPECTOR Experience", in *Artificial Intelligence Applications for Business*, W. Reitman ed., Ablex Publishing Corp., 1984

[Duffy 84]

J. Duffy, *Analysis of Mechanisms and Robot Manipulators*, Edward Arnold Co., 1984

[ElMaraghy 86]

H. ElMaraghy, "Kinematic and Geometric Modelling and Animation of Robots", in Proceedings of Graphics Interface-86, pp. 15-19

[Erman 80]

L. Erman, et al., "The Hearsay-II speech understanding system: integerating knowledge to resolve uncertainty, "Computing Surveys, Vol 2, June 1980, pp. 213-254

[Fetter 82]

W. Fetter, "A progression of human figures simulated by computer graphics", IEEE Computer Graphics and Applications, Vol. 2, pp. 9-13

[Fikes 71]

R. Fikes, "STRIPS: a new approach to the application of theorem proving to problem solving", Artificial Intelligence, Vol 2 (3), pp. 189-208

[Fleischer 84]

K. Fleisher, M. Vickers, A. Marion, J. Davis, "Towards Expressive Animation for Interactive Characters", in Proceedings Graphics Interface-84, 1984

[Forgy 77]

C. Forgy and J. McDermott, "OPS: A Domain-Independent Production System Language", Proceedings of IJCAI-77, 1977, pp. 933-939

[Ginsberg 82]

C. Ginsberg and D. Maxwell, "Graphical Marionette: A Modern-Day Pinocchio", Technical Report, Architecture Machine Group, MIT 1982

[Girard 85]

M. Girard and A. Maciejewski, "Computational Modeling For the Computer Animation of Legged Figures", Computer Graphics , Vol19:3 (1985), pp. 263-270

[Girard 87]

M. Girard, "Interactive Design of 3-D Computer-Animated Legged Animal Motion", IEEE Computer Graphics and Applications, (Special Issue), Vol.7(6), June 1987

[Halpern 85]

J. Halpern and Y. Moses, "A Guide to the Modal Logics of Knowledge and Belief", in Proceedings of IJCAI-85, 1985

[Harmon 85]

J. Harmon and D. King, *Expert Systems: Artificial Intelligence in Business*, John Wiley&Sons, 1985

[Hayes 74]

Patrick H. Hayes, "Some Problems and Non-Problems in Representation Theory", Proceedings AISB Summer Conference, University of Sussex, 1974

[Hayes 77]

Patrick H. Hayes, "In Defence of Logic", Proceedings IJCAI-77, 1977, pp. 559-565

[Herbison-Evans 78]

D. Herbison-Evans, "NUDES2: A Numeric Utility Displaying Ellipsoid Solids", Computer Graphics, Vol.12, pp. 354-356

[Hewitt 72]

C. Hewitt, *Description and theoretical analysis (using schemata) of PLANNER: A language for proving theorems and manipulating models in a robot*, Ph.D. Thesis, Dept. of Mathematics, MIT, 1972

[Horvitz 86]

Eric J. Horvitz, David E. Heckerman, Curtis P. Langlotz, "A Framework for Comparing Alternative Formalisms for Plausible Reasoning", in Proceedings of AAAI-86, 1986

[Kahn 79]

Kenneth M. Kahn, *Creation of Computer Animation from Story Descriptions*, Ph.D. Thesis, Dept. of Computer Science, M.I.T. 1979

[Kautz 82]

Henry A. Kautz, "Planning Within First-Order Dynamic Logic", in Proceedings of CSCSI- 82, 1982

[Kroyer 86]

B. Kroyer, "Animating with a Hierarchy", in Seminar on Advanced Computer Animation, Proceedings SIGGRAPH-86, pp. 266-288

[Hayes-Roth 83]

F. Hayes-Roth, D.B. Lenat, D.A. Waterman (eds.), *Building Expert Systems*, Addison-Wesley 1983

[Kochanek 82]

D. Kochanek, R. Bartels, K. Booth, "A Computer System for Smooth Keyframe Animation", Tech. Report, University of Waterloo, December 1982

[Korein 82]

James U. Korein and Norman I. Badler, "Techniques for Generating the Goal-Directed Motion of Articulated Structures", IEEE Computer Graphics and Applications, Nov. 1982, Vol.2 (9), pp. 71-81

[Lee 82]

C.S. Lee, "Robot Arm Kinematics, Dynamics and Control", IEEE Transactions on Computers, Vol. 15 (12), pp. 62-80

[Lindsay 80]

R. Lindsay, B. Buchanan, E. Feigenbaum, J. Lederberg, *Applications of Artificial Intelligence for Chemical Inference: The DENDRAL Project*, McGraw-Hill, 1980

[Lozano-Perez 80]

T. Lozano-Pérez, "Automatic Planning of Manipulator Transfer Movements", Tech. Report AI Memo 606, MIT 1980

[Lozano-Perez 83]

T. Lozano-Pérez, "Spatial Planning:A Configuration Space Approach", IEEE Transactions on Computers, Vol. 32 (2), February 1983, pp. 108-117

[McCalla 83]

G. McCalla and N. Cercone (eds.), *IEEE Computer: Special Issue on Knowledge Representation*, Vol.16 (10), October 1983

[McCarthy 80]

John McCarthy, "Circumscription-A Form on Non-Monotonic Reasoning", Artificial Intelligence , Vol.13, 1980, pp. 27-39

[McDermott 81]

J. McDermott, "R1: The Formative Years", AI Magazine, Vol 2 (2), 1981

[McDermott 82]

D. McDermott, "A temporal Logic for reasoning about processes and plans", Cognitive Science, Vol. 6, 1982, pp. 101-155

[Minsky]

M. Minsky, "A Framework for Representing Knowledge", in *The Psychology of Computer Vision*, P.H. Winston (ed.), McGraw-Hill 1975, pp. 211-277

[Nau 82]

Dana S. Nau, " Expert Computer Systems: A Tutorial", Technical Report TR-1201, Computer Science Dept., University of Maryland, College Park, Maryland, August 1982.

[O'Connor 84]

D.E. O'Connor, "Using Expert Systems to Manage Change and Complexity in Manufacturing", in Artificial Intelligence Applications for Business, W. Reitman (ed.), Ablex Publishing Corp., 1984

[Paul 81]

R. Paul, *Robot Manipulators: Mathematics, Programming and Control*, MIT Press 1981

[Reiter 78]

R. Reiter, "On Closed World Data Bases", in *Logic and Databases*, H. Gallaire and J. Minker eds., Plenum Press, 1978

[Reiter 80]

R. Reiter, "A Logic For Default Reasoning", Artificial Intelligence , Vol.13, 1980, pp. 81-132

[Ridsdale 86]

G. Ridsdale, S. Hewitt and T. Calvert, "The Interactive Specification of Human Animation", in Proceedings of Graphics Interface-86, pp. 121-130

[Robinson 65]

J. Robinson, "A machine-oriented logic based on the resolution principle", JACM, Vol.23 (2), January 1965, pp. 23-41

[Ryman 83]

R. Ryman, A. Patla and T. Calvert, "Use of Labanotion for clincal analysis of movement", in Confernce of the International Congress Kinetography Laban, August 1983

[Sacerdoti 74]

Earl D. Sacerdoti, "Planning in a Hierarchy of Abstraction Spaces", Artificial Intelligence, Vol. 5, 1974, pp. 115-135

[Sacerdoti 77]

Earl D. Sacerdoti, *A Structure for Plans and Behaviour*, Elsevier North-Holland Inc., 1977

[Seidel 81]

Raimund Seidel, "A New Method For Solving Constraint Satisfaction Problems", in Proceedings of IJCAI-81, 1981, pp.338-342

[Singh 83]

B. Singh, "A Computerized editor for Benesh movement notation", Master's Thesis, University of Waterloo, 1983

[Shank 77]

R. Shank and R. Abelson, *Scripts, Plans, Goals and Understanding*, John Wiley & Sons, 1977

[Shapiro 79]

Stuart C. Shapiro, "Numerical Quantifiers and Their Use in Reasoning with Negative Information", in Proceedings IJCAI-79, 1979, pp.791-796

[Shortliffe 76]

E.H. Shortliffe, *Computer-Based Medical Consultation: MYCIN*, American Elsevier, 1976

[Sloman 81]

Aaron Sloman and Monica Croucher, "Why Robots Will Have Emotions", in Proceedings of IJCAI-81, 1981, pp. 197-202

[Sobek 85]

Ralph Sobek, "A Robot Planning Structure Using Production Rules", in Proceedings of IJCAI-85, (1985) pp. 1103-1105

[Stefik 81]

Mark Stefik, "Planning With Constraints (MOLGEN: Part 1)", Artificial Intelligence 16 (1981), pp. 111-140

[Steketee 85]

S. Steketee and N. Badler, "Parametric keyframe interpolation incorporating kinetic adhustment and phrasing control", Computer Graphics, Vol. 19, pp. 255-262

[Stuart 85]

C. Stuart, "An Implementation of a Multi-Agent Plan Synchronizer", in Proceedings of IJCAI-85, 1985

[Sturman 84]

D. Sturman, "Interactive key frame animation of 3-D articulated models", in Proceeding Graphics Interface-84, 1984, pp. 35-40

[Sutherland 63]

I. Sutherland, "SKETCHPAD: A man-machine graphical communication system", in Procedings of IFIPS Spring Joint Conference, 1963

[Thomas 81]

Frank Thomas and Ollie Johnston, *Disney Animation—The Illusion of Life*, Abbeville Press, New York 1981

[Weber 78]

L. Weber, S. Smoliar, N. Badler, " An architecture for the simulation of human movement", in Proceedings ACM Anual Conference, 1978, pp. 737-745

[Wilensky 83]

R. Wilensky, *Planning and Understanding: A Computational Approach to Human Reasoning*, Addsion-Wesly 1983

[Wilhelms 86]

J. Wilhelms, "Virya - A Motion Control Editor for Kinematic and Dynamic Animation", in Proceedings of Graphics Interface-86, 1986, pp 141-146

[Wilhelms 87]

J. Wilhelms, "Using Dynamic Analysis for Realistic Animation of Articulated Bodies", IEEE Computer Graphics and Applications, (Special Issue), Vol.7(6), June 1987

[Winston 81]

Patrick H. Winston , "Learning New Principles from Precedents and Exercises: The Details", Technical Report AI Memo No.632, MIT AI Lab, 1981.

[Winston 84]

P.H. Winston and K.A. Prendergast (eds.), *The AI Business: The Commercial Uses of Artificial Intelligence*, MIT Press, 1984

[Woods 75]

W.A. Woods, "What's in a Link:Foundations for Semantic Networks, Representation and Understanding", in *Studies in Cognitive Science*, Academic Press, 1975

[Zadeh 83]

L.A. Zadeh, "Commonsense Knowledge Representation Based on Fuzzy Logic", IEEE Computer, Vol. 16 (10), October 1983, pp. 61-66

[Zeltzer 82]

D. Zeltzer, "Motor control techniques for figure animation", IEEE Computer Graphics and Applications, Vol 2 (9), November 1982, pp. 53-59