

PHYSICALLY BASED MECHANICAL METAPHORS
IN ARCHITECTURAL SPACE PLANNING

A Dissertation

by

SCOTT ANTHONY ARVIN

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2004

Major Subject: Architecture

© 2004

SCOTT ANTHONY ARVIN

ALL RIGHTS RESERVED

PHYSICALLY BASED MECHANICAL METAPHORS
IN ARCHITECTURAL SPACE PLANNING

A Dissertation

by

SCOTT ANTHONY ARVIN

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Approved as to style and content by:

Donald H. House
(Chair of Committee)

Mark J. Clayton
(Member)

Robert E. Johnson
(Member)

Bruce H. McCormick
(Member)

Phillip J. Tabb
(Head of Department)

May 2004

Major Subject: Architecture

ABSTRACT

Physically Based Mechanical Metaphors
in Architectural Space Planning. (May 2004)
Scott Anthony Arvin, B.Arch., University of Notre Dame;
M.S., Troy State University
Chair of Advisory Committee: Dr. Donald H. House

Physically based space planning is a means for automating the conceptual design process by applying the physics of motion to space plan elements. This methodology provides for a responsive design process, allowing a designer to easily make decisions whose consequences propagate throughout the design. It combines the speed of automated design methods with the flexibility of manual design methods, while adding a highly interactive quality and a sense of collaboration with the design.

The primary assumption is that a digital design tool based on a physics paradigm can facilitate the architectural space planning process. The hypotheses are that Newtonian dynamics can be used 1) to define mechanical metaphors to represent the elements in an architectural space plan, 2) to compute architectural space planning solutions, and 3) to interact with architectural space plans.

I show that space plan elements can be represented as physical masses, that design objectives can be represented using mechanical metaphors such as springs, repulsion fields, and screw clamps, that a layout solution can be computed by using these elements in a dynamical simulation, and that the user can interact with that solution by applying forces that are also models of the same mechanical objects. I present a prototype software application that successfully implements this approach.

A subjective evaluation of this prototype reveals that it demonstrates a feasible process for producing space plans, and that it can potentially improve the design process because of the quality of the manipulation and the enhanced opportunities for design exploration it provides to the designer.

I found that an important characteristic of this approach is that representation, computation, and interaction are all defined using the same paradigm. This contrasts with most approaches to automated space planning, where these three characteristics are usually defined in completely different ways.

Also emerging from this work is a new cognitive theory of design titled ‘dynamical design imagery,’ which proposes that the elements in a designer’s mental imagery during the act of design are dynamic in nature and act as a dynamical system, rather than as static images that are modified in a piecewise algorithmic manner.

To my wife, Laura

ACKNOWLEDGMENTS

I would like to thank my committee members, Dr. Mark Clayton, Dr. Robert Johnson, and Dr. Bruce McCormick for their contribution and guidance throughout the long process of completing this work. I would also like to thank Larry Degelman, Ergun Akleman, Karen Hillier, Keith Sylvester, Charles Graham, Bob Warden, Valerian Miranda, and Tom Parker, for a wide variety of professorial support; fellow Ph.D. students Greg Schmidt, David Beattie, and Veronica Seobarto; and fellow vizzers John Ferguson, Jean-Claude Kalache (and his wife Sue), Keith Klohn, and especially Quintin King. Quintin managed to answer every one of the technical questions I put to him, and I still think he's amazing, even if all my questions were similar in nature! I would also like to thank Kevin Glueck, the friendliest system administrator in the world; my good friend Michael Baloy; and Dave Plunkett at Autodesk, Inc. for feeding the family while I finished. Thanks to anyone else who I have forgotten but who contributed in some way to this work, who encouraged me when things looked bleak, or who got tired of hearing about it.

I would especially like to acknowledge my advisor, Dr. Donald House. I cannot imagine another advisor with the fantastic qualities that he has, and doubt very much that I would have finished without his guidance, support, and willingness to listen. He is an amazingly kind and intelligent man, who is a true ambassador of his beliefs. I aspire to live my life in the calm and gentle manner that he does. Thanks Don!

Thanks to my parents, Leo and Carolyn Arvin, for their constant love and prayers throughout my life, and for having the trust to allow me to think my own thoughts and pave my own way. And thanks to my wonderful children, Jonathan

and Catherine, who have been affected by this work in one form or another for their entire lives.

Finally, of course, I wish to give my thanks and love to my wife, Laura. Without her love and support I would definitely have given up on this thing long before it was finished! Thanks for never giving up on me!

This work was supported in part by a Lechner Fellowship, a Caudill Graduate Research Fellowship, a College of Architecture Research Grant, an O. N. Mitchell, Jr. Endowed Fellowship in Construction Management, the Visualization Laboratory, and the Department of Architecture in the College of Architecture at Texas A&M University.

TABLE OF CONTENTS

	Page
ABSTRACT	iii
DEDICATION	v
ACKNOWLEDGMENTS.....	vi
TABLE OF CONTENTS.....	viii
LIST OF TABLES	xi
LIST OF FIGURES	xii
1 INTRODUCTION	1
1.1. A physical paradigm for digital design	4
1.2. Problem statement.....	7
1.3. Methodology	8
1.4. Summary of results	9
2 LITERATURE REVIEW	12
2.1. Physically based modeling	12
2.2. Computer-aided space planning	14
2.3. Design thinking	25
2.4. Physically based methods in space planning	32
2.5. Why use physics in digital design?.....	34
3 REVIEW OF PHYSICALLY BASED MODELING.....	40
3.1. Definitions	40
3.2. Example: Thrown ball	42
3.3. Elements of a physical system	43
3.4. Summary	66
4 A PHYSICALLY BASED APPROACH TO SPACE PLANNING	68
4.1. Design elements	69
4.2. Design objectives	80

	Page
4.3. Process	110
4.4. Summary	112
5 PROTOTYPE IMPLEMENTATION	114
5.1. Implementation	114
5.2. Suggested interaction	116
5.3. Early worked example	119
5.4. Final worked example	125
5.5. Summary	128
6 COMPLEXITY ANALYSIS	129
6.1. Computational complexity	129
6.2. Space complexity	138
6.3. Summary	138
7 DISCUSSION	140
7.1. Prototype	140
7.2. General observations	150
7.3. Summary	157
8 THEORETICAL IMPLICATIONS	158
8.1. Dynamical design imagery	158
8.2. Clay argument	161
8.3. Cognitive science argument	162
8.4. Design-mind-motion argument	165
8.5. Summary	172
9 CONCLUSIONS	173
9.1. Contributions	174
9.2. Future work	175
9.3. Conclusion	184
REFERENCES	185
APPENDIX A PROTOTYPE INTERFACE	192
APPENDIX B APF FILE FORMAT	237
APPENDIX C APF SAMPLE FILE - COUNSELING CENTER	248

APPENDIX D APF SAMPLE FILE - RESIDENCE	250
APPENDIX E ALGORITHM TO UNION MULTIPLE POLYGONS.....	263
VITA	277

LIST OF TABLES

TABLE		Page
3-1	Comparison of truncated Taylor series used in integration methods . .	58
3-2	Stability depends on mass, spring, and damper constants	61
4-1	Properties of a design objective	82
4-2	Sample levels of importance	84
4-3	Classes of objectives and their properties	85
4-4	Physical analogues of design objectives	86
5-1	Implemented concepts from section 4	115
5-2	Dynamic and other constants	124
8-1	Similar Mind/Motion/Design separations	167

LIST OF FIGURES

FIGURE	Page
1-1	Simple space with forces acting on its elements 6
3-1	A ball thrown through the air 44
3-2	A mass-spring-damper system 51
3-3	Problems with Euler Integration 53
3-4	Midpoint method 56
3-5	Stability example: a 2^{nd} order linear system 59
3-6	Collision detection and response 60
3-7	Example of a dynamic constraint 64
3-8	Basic numeric simulation algorithm 67
4-1	Designer's problem 69
4-2	Dynamics problem 69
4-3	Point node 71
4-4	Line node 72
4-5	Plane node 73
4-6	An arbitrary polygonal shape represented with edge line nodes 75
4-7	A rectangle represented with vertex point nodes and edge line nodes . 76
4-8	A space plan using vertex polygonal representations 77
4-9	A space with soft edges wrapping around a space with rigid edges . . . 77
4-10	Parent space with and without a defined boundary 79

FIGURE	Page
4-11	Use of a space local coordinate system 80
4-12	Interior objectives 88
4-13	Exterior objectives 90
4-14	Orientation objectives 92
4-15	Adjacency objective 96
4-16	Separation objective 97
4-17	Area objective 100
4-18	Area objective forces 100
4-19	Proportion objective 102
4-20	Proportion objective forces 103
4-21	Alignment objective 106
4-22	Alignment objective forces 106
4-23	Topological elements vs geometric elements 109
4-24	Simulation 111
5-1	Sample Adjacency Matrix 119
5-2	Sample topological resolution 120
5-3	Sample geometric resolutions 122
5-4	Early worked example 123
5-5	Final worked example - from initial placed positions 125
5-6	Final worked example - after manual rearrangement of spaces, and doing topological and geometric resolutions 126
5-7	Final worked example - after manual rearrangement of spaces, and doing only geometric resolution 127

FIGURE	Page
6-1	Simplified algorithm to integrate over a set of nodes 131
6-2	Simplified algorithm to find and handle contacts between spaces 133
6-3	Simplified algorithm to compute union of space polygons 134
6-4	Algorithm to compute state energy 136
6-5	Algorithm to perform one time step 137
8-1	Mental rotation 161
8-2	Dynamical Design Imagery - the three major concepts of Design, Mind, and Motion 166
8-3	Dynamical Design Imagery - some characteristics of the three major concepts 169
8-4	Dynamical Design Imagery - theories as combinations of two ma- jor concepts 170
8-5	Dynamical Design Imagery - a combination of all three major concepts 171

1. INTRODUCTION

Many approaches to automated architectural design incorporate much building and construction knowledge. Nonetheless, architects sometimes avoid them because they do not provide the freedom of easy design manipulation and exploration. Thus, most architects tend to use tedious manual means for creating their designs. In contrast, architects sometimes use terms such as “manipulate,” “mold,” and “massage” when describing their interaction with design elements during the early stages of design. These words evoke images of spaces as pieces of clay being sculpted by the application of forces to achieve the designer’s objectives. A design is dynamically transformed under the influence of these forces. Within this metaphor forces represent design objectives, and the process of resolving a set of forces represents the creation of a design. Once design is thought of in this force-based metaphor, physically based simulation becomes a potential methodology for realizing a computer aided design process. This dissertation proposes this approach to bridge the gap between current automated design methods that provide inadequate interaction with the designer, and current manual design methods that require too much tedious work, bringing together the benefits of both.

Imagine an architectural design that is responsive, that feels alive, that continually responds to the changing decisions of the designer. Most objectives specified in an architectural program affect the position, size and shape of some building element in space. These objectives are typically interconnected, such that changing one objective causes changes in the application of others. A responsive building model would take these changes into account, continually adjusting to the current state of

The journal model is *Environment and Planning B: Planning and Design*.

unfulfilled objectives to produce a design that tries to meet those objectives. For example, if the designer moves the location of a wall, not only would elements attached to the wall get updated, but the rooms on each side should change area as well. Those rooms can automatically adjust to maintain their correct area, but these changes may cause other dimensions to be incorrect. A responsive design process would allow the designer to specify design decisions while automatically propagating the consequences of those decisions to the rest of the design.

The architectural design problem can be loosely defined as the process of creating a final building design from the objectives initially specified in an architectural program. Non-trivial architectural design problems are ill-defined (Yoon, 1992, p. 8) and over-constrained. They are ill-defined in that initial design specifications are incomplete. Additional design knowledge needs to be added throughout the design process in order to arrive at a final solution. This knowledge can be added either manually by the designer or automatically by a computer application. Architectural design problems also tend to be over-constrained in that initial specifications are often in conflict with each other, and therefore no final design exists that meets *all* design objectives. Over-constrained problems tend to have many almost correct solutions, and need some method of finding the ones that are most correct (Balachandran and Gero, 1987).

Architectural design is a highly interactive process. The final product of a design is more dependent on the *process* of design than it is on a well defined list of initial design objectives. Much has been written about how architects employ visual and graphic thinking in their design process (Arnheim, 1969; Laseau, 1980; McKim, 1972). Scriabin and Vergin (1975) note that

... attempts to use fully automated computer algorithms to solve the lay-

out problem should be reexamined with a view of incorporating man's visual capability into the procedures, especially since the real layout problem involves many factors which cannot readily be incorporated into a computer program, but which a man can take into account while designing a layout.

Any method of automating the architectural design process needs to accommodate the three fundamental characteristics just described: by 1) modifying sets of design objectives in an ill-defined problem, 2) searching for a good solution in an over-constrained problem, and 3) giving the designer a very high level of interaction with the design thereby enabling active and visual exploration of the design space.

I propose a responsive design approach to architectural space planning that uses a physically based metaphor for the early stages of the conceptual design process. This approach addresses the ill-defined nature of the problem by allowing designers to add and modify a set of force-based design objectives throughout the design process. Searching for a solution in an over-constrained problem can be done either automatically or manually. This new approach provides an automated component to this search by using physically based simulation on appropriately represented building elements, and introduces a manual component by implementing it within a computational framework that provides for extensive real time user interaction. Thus, the designer is included as a necessary and integral part of the 'search' process.

The significant benefit to using this physically based responsive design approach should be that the *quality of interaction* with a design is enhanced. The design should feel alive to the designer and promote a sense of collaboration between the designer and the design. The *quality of design results* should be enhanced, not because of an improvement in automation, but because of an improvement in design interaction.

1.1. A physical paradigm for digital design

The initial idea for a physically based approach to space planning came from thoughts about how one could design an interface for specifying architectural programs. Adjacency requirements between spaces are typically specified in a matrix by assigning each space pair a descriptive term, such as immediate, important, convenient, or unimportant (Karlen, 1993). I thought the same task might be done numerically in a graphical user interface by displaying a slider between two space centers, and adjusting the slider based on the strength of the adjacency. I then made a conceptual leap and replaced the slider with a virtual spring, a basic element used in physically based modeling and animation. The spring could apply forces to the spaces causing them to move toward each other. The applied force would be proportional to the strength of the adjacency. From here the natural next step is to define a system of such springs connecting spaces, and to use this system to aid in producing space plans. Once such a system exists, other architectural design objectives can be represented using similar mechanical metaphors.

Physically based modeling, a sub-field of computer graphics and visualization, attempts to represent motion and changes in geometry by modeling objects as mechanical elements that behave according to the laws of physics. Dynamics are most often derived by the use of forward numerical simulation over discrete time intervals. In a forward simulation, the system is moved from its state at some current time to its state at the next discrete time step, using forces to determine accelerations, and thus changes in velocity, during the time step, and velocities to determine translations. The process of making this forward extrapolation is called numerical integration. The elementary concepts of physically based modeling are reviewed in more detail in section 3, “Review of Physically Based Modeling.”

A physically based approach to automated architectural space planning should be valid for the following reasons: 1) It is very easy to think of architectural design objectives in terms of forces being applied to architectural elements (Alexander (1964) uses a wide variety of physical terms in his discussions about the design process); 2) The process for solving a physically based system through numerical integration is a holistic one, progressing from one time step to the next while accounting for all forces acting on all elements of the system at the same time; and, 3) objects that follow the laws of physics appear natural to humans, since we continually experience physical interaction in the world. Human interaction with these simulated objects should therefore feel natural and intuitive.

A particle system is a classical physically based modeling technique where the motion and display of a number of point particles are used to simulate existing or imaginary phenomena. Each particle that exists in a simulated environment has a position, mass, and velocity; it interacts with ‘physical’ elements or other particles in its environment; and it has a lifespan during which it exists. The characteristics of its environment, such as gravity strength and direction, and air viscosity, are specified, and a motion simulation is run. Particle systems can be used to simulate such natural phenomena as waterfalls and fire.

The method of physically based space planning proposed here is essentially a classical particle system with springs applied to the specific problem domain of architectural space planning. In applying physically based techniques to space planning, the first problem is how to represent the elements of a space plan, the spaces and the walls surrounding them, such that forces can be made to act on them. The second problem is how to represent different architectural design objectives as forces that can be applied to these elements. It is useful to think of design objectives as wants or needs. For example, space *A* “wants” to be next to space *B*, or space *C* “needs” to

be 200 square feet in area. It then becomes easier to choose an applicable mechanical metaphor to define the physical forces needed to accomplish the design objectives. Figure 1-1 shows a simple space with forces acting on its elements. Arrows labeled a represent forces applied to the space location, which may change the way this space relates to another. Arrows labeled b represent forces applied to the polygonal edges of the space boundary, which may change the geometric position of the edges.

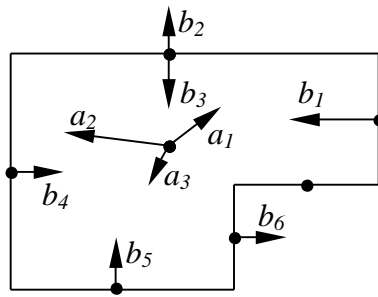


Fig. 1-1. Simple space with forces acting on its elements

Physically based methods are computationally expensive, so using them to solve a simple space layout problem may seem at first like using a sledge hammer to drive a finishing nail. Why employ a complex method such as physics equations to solve a seemingly simple problem? As I will show, the benefit of their use is not so much that they aid in the solving of the problem, but that they add a ‘quality’ to the design process that few other methods can. With recent advances in computational capability, the application of physics to a variety of problem domains has become more and more possible.

In an article reporting on advances in solving satisfaction problems, Peterson (2000) describes how the principles of physics can shed some light on the nature of

those problems. It turns out that satisfaction problems appear to have characteristics of phase transitions similar to that of some types of matter. A satisfaction problem with few constraints has many solutions, while one with too many constraints is insoluble. The difficulty lies in the middle ground, where if the number of constraints approaches a critical number it becomes difficult to tell if the problem is soluble or not. For some problems the transition from soluble to insoluble is discontinuous, similar to the transition when water freezes, while in others the transition is continuous. Peterson quotes Lenore Blum - “Looking at a problem from a new perspective can add new insight. The idea of introducing the concept of phase transitions into [computational] complexity has this potential. Even if it does not characterize complexity, figuring out to what extent it may or may not be relevant would be extremely interesting.” (Peterson, 2000) Looking at problems in physical terms can shed some light on their nature, and be used to aid in solving them.

1.2. Problem statement

The assumption, then, is that *a digital design tool based on a physics paradigm can facilitate the architectural space planning process, and possibly any dimensional design process.* This assumption leads to the hypotheses addressed here: Newtonian dynamics can be used

1. to define mechanical metaphors to represent the elements in an architectural space plan,
2. to compute architectural space planning solutions, and
3. to interact with architectural space plans.

Representation, computation, and interaction can all be encompassed within the same dynamical system.

Two individual questions are raised by these hypotheses. The first question is one of representation: How can the elements in a space planning design problem be modeled with mechanical analogues? Given this representation, the second question is one of implementation: How does an implementation of this representational model work?

1.3. Methodology

The representation of space planning elements will be addressed by applying a variety of common techniques used in physically based modeling to the elements of a space plan, as introduced earlier. Given this representation, an algorithm will be defined to produce an instance of a space plan given a dimensional state and a number of design objectives, and to provide a way for designers to manipulate a plan by modifying existing objectives and adding new ones. This representation, computation and interaction of architectural space plan elements as mechanical analogues is presented in section 4, “A physically based approach to space planning.”

A prototype computer application will be developed using this representational model. The scope of this implementation will be limited to those architectural space plans consisting of rectangular shaped spaces, and single story buildings with single height spaces. Research in space planning approaches is often limited to rectangles in order to simplify the development of new ideas, such as that presented by Liggett and Mitchell (1981b), Flemming (1978), and Akin et al. (1988). I recognize that a truly useful approach to design needs to be able to handle more complex buildings including non-rectangular shapes, multi-storied buildings, spaces with different heights, and

other design objectives. However, if the concepts of physically based space planning can be validated for basic design types, it should be possible to extend them in the future to handle more complex real-world design problems. The implementation of the prototype computer application and some worked examples are presented in section 5, “Prototype implementation,” and an analysis of the computational and space complexity of the implementation is presented in section 6, “Complexity analysis.”

The resulting prototype implementation will be used to develop a plan for a simple building. Analyzing its performance in this process, I will attempt to answer such questions as “does it behave as initially visualized?,” “does it provide a compelling design experience?,” “does it aid the design process?,” “is it ‘fun’ to use?,” and “how can it be improved?” Some general observations about the concepts presented, their implementation, and the potential usefulness of the proposed approach in the architectural space planning process are made in section 7, “Discussion.”

1.4. Summary of results

The results and contributions of this research are highlighted below. They will be revisited in section 9, “Conclusions,” along with suggestions for future work.

1.4.1. Representation

The elements of an architectural space plan were successfully modeled as physically based elements. Spaces and walls were represented as masses on which forces can be applied. A number of design objectives were represented as force applicators, using such mechanical metaphors as springs, repulsion fields, balloons, clay, and screw clamps.

1.4.2. Implementation

A prototype software application was successfully implemented. Using the physically based architectural representations mentioned above, and numerical simulation of a physical environment, space planning solutions could be computed. The force based design objectives moved design spaces and walls from one design state to another.

1.4.3. Interaction

A key realization that emerged from working with the prototype application was the importance of the quality of user interaction with the design. What started out as a method for automatically producing a space plan from a set of initial objectives turned into a method of interacting with the space plan itself. Since the physical objects we interact with daily behave according to intuitively familiar physical laws, and the elements in a physically based space plan behave according to similar laws, the interaction with those elements seems to produce a more satisfactory design experience.

1.4.4. Contributions

The primary contribution of this work is a mapping from architectural space planning concepts to physically based metaphors of mechanical objects. The proof of the success of this mapping is a working software application, and a demonstration of its use in creating a sample architectural floor plan.

1.4.5. Theoretical Implications

An additional contribution that emerged from this work is a cognitive theory of design titled ‘dynamical design imagery,’ which will be discussed in greater detail

in section 8, “Theoretical Implications”. This interesting though highly speculative theory proposes that the elements in a designer’s mental imagery during the act of design are dynamic in nature and act as a dynamical system. Although it will be difficult to gather empirical evidence, if it does turn out to have some validity it could have significant implications for the implementation of future digital design tools.

I will now review in section 2 some of the literature in computer-aided space planning, physically based modeling, and design thinking, as well as some emerging ideas about the role of dynamics in cognitive science. This will be followed in section 3 by an overview of the basic techniques used in physically based modeling, which will be necessary in order to understand their application to architectural space planning.

2. LITERATURE REVIEW

The two main fields of research related to this dissertation are computer-aided space planning and physically based modeling. However, it cannot be neatly classified into only these two areas, and also relates significantly to other major fields such as human-computer interaction, design thinking, and cognitive science. A thorough review of the literature in each of these fields is not practical, so this section describes only some of the important research in each field, and some of the previous research directly related to ideas presented in this dissertation.

2.1. Physically based modeling

Physically based modeling is a sub-field of computer graphics and visualization. It attempts to represent dynamic motion and changes in geometry by modeling objects as mechanical elements that behave according to the laws of physics. Dynamics are most often derived by the use of forward numerical simulation over discrete time intervals. In a forward simulation, the system is moved from its state at the current time to its state at the next discrete time step, using forces to determine accelerations, and thus changes in velocity during the time step, and velocities to determine translations. The process of making this forward extrapolation is called numerical integration. An excellent introduction to the concepts of physically based modeling and a practical guide to the implementation of these concepts in the computer is given in Witkin and Baraff (1997).

This research does not extend the field and techniques of physically based modeling, but instead applies those techniques in a new way to the field of architectural space planning. An extensive review of the fundamental techniques of physically based modeling used is given in section 3, “Review of physically based modeling.”

One basic use of physically based modeling is in particle systems, introduced by Reeves (1983), in which a collection of particles, each of which has a mass, position, velocity, and display properties, are made to appear as desired physical objects such as water in a waterfall.

A spring is one of the fundamental objects introduced later to model space planning design objectives. Examples of using springs to model the behavior of flexible objects include Haumann (1987) and Haumann and Parent (1988).

Other techniques in physically based modeling that are relevant to this research and will be mentioned later include constraining object movement relative to other objects (Witkin and Kass, 1988; Barzel and Barr, 1988), interacting with dynamic models (Witkin et al., 1990; Witkin and Welch, 1990), and collision detection between objects (Moore and Wilhelms, 1988).

Physically based modeling has been used to model the realistic behavior of rigid bodies (Barzel and Barr, 1988; Baraff, 1989), deformable models (Terzopoulos et al., 1987), and flocking behavior of a large number of objects (Reynolds, 1987).

House and Kocmoud (1998) use physically based modeling techniques to create what they call ‘continuous cartograms’. A cartogram is a map that displays data about a region by, for example, coloring or shading the parts of the region based on data values for those parts. For example, the states in a map of the United States can be colored red or blue based on presidential voting for a specific year. One problem with cartograms such as this is that the percent area covered with one color may not represent the related percentage of the data value. An area cartogram scales the regions so that their displayed area correlates with the data being represented. House and Kocmoud use physically based techniques to perform this scaling. Springs of various types are used to maintain the length and angle of region boundaries relative to each other, and to apply forces to make the regions become larger or smaller over

time so as to create an area cartogram that is more recognizable than those created with other methods. This task can be thought of as a highly restrictive subset of a floor planning problem - ‘given floor plan (map) A , produce another floor plan (similarly recognizable map) B , where the area of each room (state or country) is proportional to the floor plan’s area as is a desired room area (data value assigned to a state) is proportional to the floor plan’s area (sum of the data values for all states).’

Physically based dynamics have recently begun to be used in geometric design. Qin (Qin et al., 1998; Qin and Terzopoulos, 1995; Qin, 1998) and Mandal et al. (1997) use physically based techniques to manipulate smooth surfaces of arbitrary topology interactively. In their approach, a user defines the points of an initial control mesh, which are manipulated by applying synthesized forces until the desired shape is achieved. In describing his physically based approach to modifying free-form deformable models Qin uses phrases that fit very well with the purpose of the research proposed here, such as “interactive design environment,” “integrates traditional design principles with advanced physics-based design techniques,” “explore and develop flexible, efficient design tools,” and “physics-based force sculpting tools for direct manipulation of geometric primitives.”

2.2. Computer-aided space planning

Methods to automate the process of architectural space planning have been studied quite heavily over the past four decades. It is a natural place for architectural researchers to begin to apply the technology of computer science because space planning is such a fundamental part of the architectural design process, and seems to be a process that lends itself to automation. As with many problems that at first glance

look easy to solve, automated space planning has turned out to be quite a complex problem.

2.2.1. Constructive placement methods

Constructive placement methods for automated space planning place one space in a building area, then iteratively place each additional space in the building in relation to those previously placed. Objective functions are used to determine the order in which spaces are placed, as well as the placement of each space.

Liggett and Mitchell (1981b) present one such method for accomplishing automated and interactive optimized space planning. First they present a way of calculating an objective function that provides a quantitative measure of the cost of locating a particular area in a building. The objective function takes into account the fixed costs of an activity, such as rent; the interactive costs of an activity relative to other activities, such as adjacency relationships; and the move costs that would be incurred from relocating an activity from one place to another. Then they present a method for creating a space layout by incrementally adding an area to a building layout based on minimizing the cost in the objective function. They do this through the method of quadratic assignment. A set of areas is mapped into a set of locations, considering the cost of assigning an area to a location, the cost of interaction between areas, and the distance between areas. The quadratic assignment method tries to find a solution, from within the set of all possible solutions, that optimizes the objective function. Since the problem is highly complex, an exact objective function cannot be calculated, so they use probability theory to calculate the expected value of the objective function based on possible future space assignments. Their method is capable of handling multi-stage designs, is zone based, and outputs a character-based graphic representation of a floor plan (Liggett and Mitchell, 1981a).

A strength of their method is that it is able to automatically produce a near optimal space layout. It is not necessary to find a globally optimum solution in architectural design because “the task is not to find one least-costly plan, but to integrate an understanding of the cost consequences of location decisions, in a well-structured way, into a search for a solution that responds to a broad spectrum of complex and often ill-defined criteria.” (Liggett and Mitchell, 1981a, p. 296) A weakness with their method is the combinatorial complexity of quadratic assignment problems, which “belong to a class of mathematical problems known as NP-complete.” (Liggett and Mitchell, 1981b, p. 282) As additional terms are added to a problem, the solution depends on all the previous terms. This algorithm then has a computational complexity of order $O(n!)$, and it is generally impossible to prove that a solution is the optimal solution. However, the space planning problem itself is not this complex. As a new space is added to a problem, it tends not to be functionally related to all other spaces, but just a subset of them, so it is more likely to be of order $O(n^2)$. The quadratic assignment approach is thus adding complexity to the problem that is not really there. This is usually not considered much of a limitation in space planning problems, because of the size of these problems relative to other NP-complete problems, and because finding *the* optimal floor plan is not of primary concern.

One of the most notable and extensive research projects in recent years is the “Software Environment to support the Early phases in building Design,” or SEED (Flemming and Chien, 1995). SEED partitions the schematic design problem into a variety of modules, one of which is SEED-Layout (Flemming and Chien, 1995). SEED-Layout supports design space exploration through an iterative constructive placement, constraint based approach that can be either manual or automated. It also supports a case-based approach where previous designs can be used to produce new designs.

Akin et al. (1988) also use a type of constructive initial placement method, which will be described later in this section in the context of design thinking.

2.2.2. *Generative methods*

Generative space planning methods seek to produce all or a large number of the possible designs within a design space. Two common methods are shape grammars and genetic algorithms.

2.2.2.1. *Shape grammars*

Shape grammars were developed with the intent of defining a *language of design*, and are an application of *phrase structure grammars* introduced by Chomsky (2003) and used to develop a theory of linguistics. They were introduced into the design domain by Stiny and Gips (1972). Whereas linguistic phrase grammars operate on alphabets to create sentences, shape grammars operate on shapes to create geometric designs. Shape grammar elements consist of individual shapes, similar to words, a vocabulary of a set of shapes, and re-write rules where one shape or set of shapes is transformed into another shape or set of shapes.

A shape grammar system can be used to generate designs, depending on the application of the re-write rules. They have been used to attempt to define a design language of ‘well-formed’ design styles such as the windows of Frank Lloyd Wright (Rollo, 1995), the architecture of Palladio (Stiny and Mitchell, 1978), and Queen Anne houses (Flemming, 1987), as well as in solid modeling (Heisserman, 1994), and urban layouts Grimsdale and Chang (1996). Shape grammars have also been used with rectangular dissections, described in more detail in the next section. Harada (1997) describes a physically based method for improving design exploration of an instance of a shape grammar design, which will be described in detail in section 2.4.

2.2.2.2. *Rectangular dissections*

A rectangular dissection is a rectangle that has been partitioned into a number of smaller rectangles such that none of the smaller rectangles overlaps each other, and no space inside the outer rectangle does not belong to an inner rectangle. A typical method for creating a rectangular dissection is to take the outer rectangle, draw one or more horizontal (vertical) lines between opposite edges, then draw one or more vertical (horizontal) lines that do not cross but whose endpoints fall on horizontal (vertical) lines, and continue to alternate between vertical and horizontal lines. Although rectangular dissections have limitations, they have provided a useful tool to developing a number of theories of automated space planning.

Flemming (1978) describes a method for automated space allocation that applies wall representations to rectangular dissections. A wall representation is a string that specifies whether a wall is horizontal or vertical, and identifies the spaces that lay on each side of the wall. A series of wall representations can then define a specific rectangular dissection. He defines a set of string substitution rules that transform one wall representation containing n spaces to another wall representation containing $n+1$ spaces that remains valid within the bounds of a set of constraints specified in a design problem. His approach searches for a set of possible design solutions by starting with a single space and continually adding additional spaces that meet specified constraints. The resulting set of solutions is then ordered based on an objective function. First, potentially all topologically feasible solutions are enumerated, and then dimensional constraints such as area are applied. He notes that most design problems are under constrained, in that there are a large number of possible solutions that meet all of the initial constraints, and that a candidate design solution need not be optimized, but need only be ‘reasonably dimensioned.’

Gilleard (1978) has a similar goal of enumerating the floor plan possibilities for a rectangular dissection. Given a planar adjacency graph with nodes representing spaces and edges representing adjacencies between spaces, its dual is created, whose edges represent walls separating spaces and whose nodes represent wall intersections. At this point, the edges of the dual graph are rarely orthogonal to the principal axes. Gilleard's approach is to specify the orientation of each dual edge as horizontal or vertical through the application of a sequence of rules, thus transforming the dual graph into a rectangular dissection. Many of the orientations are unique, but some have multiple possibilities. The enumeration of the possible rectangular dissections based on these non-unique edges is the significant contribution of his approach.

2.2.2.3. Genetic algorithms

Genetic algorithms are a class of methods that change a data set through rules of variation and selection. Specific solutions to a problem are evolved through the use of these rules, and then compared to some fitness function to see if any of the new solutions are better than the old one. Gero and others have recently applied concepts from genetic algorithms in their work on evolutionary approaches to space planning (Gero, 1998; Gero and Kazakov, 1998; Jo and Gero, 1998).

The basic concepts used in genetic algorithms are the genotype, the phenotype, expression, selection, and reproduction (Sims, 1991, p. 319). The *genotype* is the genetic information that can be used to describe a specific individual, while the *phenotype* is the individual itself. *Expression* is the process of realizing the phenotype from the genotype; a genotype expresses a phenotype. Sims (1991) uses genetic algorithms to create digital images, in which the genotype is a symbolic expression whose result is a color value at a specific pixel location, and the phenotype is an image created with that expression. *Selection* is the act of determining which of

the expressed phenotypes survives to the next generation. The selection process is done through some measurement of fitness of a phenotype, which can be done by a person as well as an algorithm. Sims uses human interaction as the fitness test in his implementation of genetic algorithms. At each generation, a number of phenotypes are presented to the user, who picks the best ones to use as the starting point for the next generation. *Reproduction* is the process of creating new genotypes from existing genotypes. Variation needs to be introduced into succeeding generations in order for evolution to occur. This can be done through *mutation*, which is randomly changing parts of a single genotype, or through *mating*, which is combining the parts of two genotypes together.

An interesting characteristic of genetic algorithms is that they can provide for the creation of new types of *beings*, such as Sims' evolving virtual creatures (Sims, 1994). The genetic code can be made to combine in such a way as to create new codes, often unimaginable to the user. Although applicable in some domains, this would typically not be useful in the space layout problem, where the number, size, and kind of spaces are well specified before design starts. Any change to the specifications results in an undesirable redefinition of the problem.

2.2.3. *Constraint methods*

Constraint methods began with the classic work of Ivan Sutherland and his SKETCH-PAD system (Sutherland, 1963). Along with the innovative use of hardware such as the light pen and buttons for selecting objects on a screen, and data structure innovations such as linked lists, Sutherland developed many user interaction techniques that form the basis of today's graphical user interfaces and direct manipulation techniques. His work in drawing precise geometric objects based on imprecise user input forms the basis of many constraint systems to this day.

Constraints have been used in architectural design (Gross et al., 1987) in, for example, three-dimensional solid modeling (Tobin, 1991; Martini, 1995), and space layout planning (Yoon, 1992). Others have described a variety of design constraint types such as dimensional, ratio, adjacency, orientation, and shape constraints (Pfefferkorn, 1975, p. 430),(Mitchell, 1977).

Papper et al. (1991) use constraints to create an interface where users manipulate virtual objects in a manner similar to the way objects are constrained in the real world. For example, when a user moves a table in a room it stops moving when it ‘hits’ a wall, or when a user places an object such as a computer over a table, the computer ‘falls’ until it rests on the table. An example of how expert test subjects used these constraints is when “they use one block as a pusher block to move several objects in the same mode,” (Papper et al., 1991, p. 219), which is similar in functionality to the alignment objective I will describe in section 4.2.7.1. They define ‘physical’ constraints such as gravity, friction, and pushing. However, constraint transformations are computed using a standard algebraic constraint solver, rather than through the use of physics as proposed here. For example, a friction constraint is defined such that if object A is on top of and touching object B and B is moved, then A is moved by the same amount. Actual friction, “the force that resists relative motion between two bodies in contact” (Merriam-Webster, 1995), plays no part in the computation of the constraint, and if in the future the application provided haptic feedback there would be no indication of the effort required to slide a heavy object across another. Instead, the friction constraint is more like an ‘on’ constraint or a ‘hierarchical position’ constraint. The extensive use of this physical terminology does suggest, however, that a constraint system that *is* physically based might be worth investigating.

Weinzapfel and Handel (1975) and Johnson et al. (1970) describe a constraint

based approach to automated space planning that is conceptually very similar to the one proposed here, in which a design problem consists of a set of spaces and a set of relationships describing constraints on the spaces. Their algorithm repeatedly iterates through each space, testing all relationships that affect a space to determine the next location that best meets those relationships. If two spaces that need to be adjacent to each other have been picked for evaluation, the new position of each space will be computed algebraically in one computational step, after which the adjacency constraint will be met. Conflicting relationships are solved using an optimization technique called Least Mean Squares Fit. A drawback of their approach is that the order in which the spaces are evaluated affects the final solution. In contrast, in the physically based space planning approach proposed here 1) *all* defined design objectives are computed in one computational step, 2) constraints are not achieved in that one step but require a number of steps that simulate the passage of time and move constraints incrementally closer to valid relationships, finally, 3) since the simulation solves for the affect of all objectives at the same time and there is no specified order of evaluation, a specific design state with a specific set of design objectives will always produce the same solution.

2.2.3.1. Objectives vs. constraints

Lawson (1997, p. 92) says that constraints “establish relationships between elements of the object being designed.” Much of the literature related to space planning uses the term *constraint* in a similar fashion. There are two ways to use constraints, as constraint achievers or as constraint maintainers. Given two elements that currently *are not* in a valid constrained relationship, a *constraint achiever* has the capability of transforming one or both elements so that they *are* in a valid relationship. In contrast, given two elements that currently *are* in a valid constrained relationship,

a *constraint maintainer* makes sure that they *remain* in that valid relationship. A constraint maintainer cannot achieve constraints, and a constraint achiever cannot maintain constraints.

A *design objective*, as used here, is really a type of constraint, specifically a constraint achiever. The term *objective* is used instead of *constraint* so as not to be confused with *dynamic constraint* which is often used in physically based modeling. Dynamic constraints are constraint maintainers. They are described later in section 3.3.7, and will be mentioned again in section 4.3.2 as the means used to maintain the distance between spaces that have collided with each other.

A number of design objectives will be defined in section 4.2, all intending to be constraint achievers in that it is desired that they transform elements from a non-valid to a valid relationship. Some, such as an adjacency objective, are defined as what might be called soft constraints, because they might not be able to successfully create a valid relationship due to conflicting objectives, a state typically described as over-constrained. Others, such as an alignment objective, are defined as what might be called hard constraints, because they will almost always be able to create a valid relationship (except in badly defined or degenerate cases). A soft constraint that rarely produces a valid relationship would be considered a problem in typical constraint based systems. However, it may turn out to have its advantages, if they can be related in some way to the following distinction between constraints and criteria described by Negroponte.

A criterion is a target, usually defined without a numeric value and described “as a direction with *-est*: smallest, widest,” etc. A constraint is a limit, usually defined with a numeric value and “being a bound delimited by *-er*: greater than, cheaper than,” etc. (Negroponte, 1975, p. 173) Negroponte notes that “as soon as there is more than one criterion, the issue becomes messy because it is necessary

to relate criteria to each other,” and that one way to define this relationship is to rephrase the problem statement, “making one of the criteria into a constraint.” The interesting statement he then makes is that

it is precisely because of this practice of forever making criteria into constraints that automated space planning yields distorted and unproductive results. While it facilitates computer programming and while it conveniently removes context, the continual rephrasing of criteria into constraints disregards all circumstances where a good solution can be found fractionally beyond one (usually arbitrarily set) limit (Negroponte, 1975, p. 173).

This statement was made almost thirty years ago, and constraint methods have no doubt advanced since then, but they are rarely used in practice and it could be argued that they still produce “distorted and unproductive results.” Design objectives that are soft constraints still fit Negroponte’s definition of constraint, but the fact that they are soft and ‘fuzzy’ suggests that they might be better described as criteria.

2.2.4. Optimization methods

Optimization is the process of finding the best solution or solutions to a problem given a number of requirements. The constructive placement method used by Liggett and Mitchell (1981b) and described previously uses quadratic assignment to find near optimal space plans.

Some optimization methods employ an iterative improvement strategy, which starts with a system in a known configuration, then steps through a number of states attempting to improve the solution at each step. Genetic algorithms, described previously, are one such iterative improvement strategy. At each step a single parameter

of the system is changed by some amount, and then an objective function for the system is calculated. If the value for the function shows that the new system is better than the old system, the new one is kept; otherwise the old one is kept. The algorithm keeps running until the objective function doesn't improve after a specified number of times. One problem with this strategy is that it tends to get caught on local optima rather than continuing until it finds the global optimum.

Simulated annealing is an iterative improvement method that introduces an *annealing schedule of temperatures* to solve the problem of getting stuck on local optima. A high temperature corresponds to a large change in the value of a system parameter, while a low temperature corresponds to a small change. The annealing schedule determines the rate at which the temperature is reduced. At each temperature, changes are continually made until the system has reached a steady state, that is, the objective function no longer improves. The temperature is reduced in slow stages until the system *freezes*. At any step, if the objective function for the new configuration is better, the new configuration is kept. If it is worse, then a probability is calculated to determine if the new worse system is kept or the system is returned to its old state, and a new step is begun. By keeping worse systems every now and then, it is possible to back out of local optima.

Gero and Kazakov (1998) note that Wilhelm and Ward (1987) have applied simulated annealing to the space planning problem as a way to solve combinatorial optimization in quadratic assignment problems.

2.3. Design thinking

A general definition of the term *design* is “to create, fashion, execute, or construct according to plan” (Merriam-Webster, 1995), which can be applied to a wide range of

activities. Similarly, Schön notes that “Herbert Simon and others have suggested that all occupations engaged in converting actual to preferred situations are concerned with design. Increasingly there has been a tendency to think of policies, institutions, and behavior itself, as objects of design” Schön (1983, p. 77). However, *design* will be used here in the more limited context of “the creation of a representation of an object which meets a set of requirements” (Woodbury, 1987, p. 13), where the representation contains geometric elements that have dimensional properties.

Arnheim (1969), in developing the idea of *visual thinking*, notes that thinking is not a process that takes place solely in the mind, but is one that includes all of the senses. What a person sees and feels has as much to do with the process of thinking as does what goes on in the brain. Thinking is a sensory experience, not simply a logical one.

Laseau (1980) takes visual thinking one step further to *graphic thinking*, where the act of sketching becomes an integral part of the cognitive process. Graphic thinking is a communication process with ourselves, in which the drawing and the act of drawing suggest new ideas. The graphic thinking process contains four parts, the eye, the brain, the hand, and the sketch, all of which “have the capability to add, subtract, or modify the information that is being passed through the communication loop.” (Laseau, 1980, p. 9) Visual artifacts enhance graphic thinking by externalizing part of the cognitive process, through the creation of objects that have their own existence. Laseau says externalized graphic thinking has the following advantages over internalized thought (quoted from Arnheim (Arnheim, 1966, p. 206)):

First, direct sensory involvement with materials provides sensory nourishment - literally ‘food for thought.’ Second, thinking by manipulating an actual structure permits serendipity - the happy accident, the unex-

pected discovery. Third, thinking in the direct context of sight, touch, and motion engenders a sense of immediacy, actuality and action. Finally, the externalized thought structure provides an object for critical contemplation as well as a visible form that can be shared with a colleague ... (Laseau, 1980, p. 11)

These advantageous characteristics of graphic thinking might be used to evaluate the usefulness of computer-aided design methodologies that propose to support the design process.

Schön (1983) introduces the concept of “reflection-in-action” in his research on how professionals think while they are performing their tasks, which has added a new dimension to our understanding of the design process. He proposes that when professionals are attempting to solve their domain specific problems, they do not simply ‘arrive’ at a solution, but are engaged in a continual and reciprocal interaction with the elements of the problem. The professional and the problem collaborate with and, in his words, “continually talk-back” to each other. The interplay between problem and solver is nicely summed up when he notes that “the unique and uncertain situation comes to be understood through the attempt to change it, and changed through the attempt to understand it” (Schön, 1983, p. 132).

In his research Schön closely studied a number of professions, one of which is the design profession. Specifically, using protocol analysis he recorded the interaction between a student and an instructor in an architectural design studio. When discussing design changes with the student the instructor often used “spatial-action language”, such as “this room might go over here”, which usually could not be understood by analyzing either the speech or the drawing alone, but could only be understood by analyzing the speech directly in the context of the drawing. This extensive use of

spatial-action language suggests a potential benefit from using a highly interactive approach to design manipulation. “Move this room over here” has a strongly physical connotation, because that’s what we do when we interact with our environment. With static drawings we are limited to *speaking* about a desired change, and *imagining* the impact that change might have on related aspects of the design. By using a physically based manipulative design tool, the designer can *act* directly on the object as if it were real, much like moving things around in our physical world, and then quickly *see* the consequences of that action.

A common concept in the discussion of design processes is the notion of a *design space* or a *problem space*, which is the collection or space of all possible solutions for a single problem with a set of specified requirements. If the requirements for a problem are modified, then the design space is also modified. A single instance or embodiment of a design solution within a design space is called a *design state*. Much research into the design process, and in creating tools to support it, has been done in the area of *design exploration*, which is *how* designers go about moving from one design state to another within a design space. Design as a process of exploration is readily apparent in Schön’s protocols. His theory of reflection-in-action is an outgrowth of this characteristic, which can be seen when he says that designers “are likely to find new and unexpected meanings in the changes they produce and to redirect their moves in response to such discoveries. And if they are good designers, they will reflect-in-action on the situation’s back-talk, shifting stance as they do so from “what if?” to recognition of implications, from involvement in the unit to consideration of the total, and from exploration to commitment” (Schön, 1983, p. 103).

Design fixation occurs when a designer gets ‘stuck’ at a specific design state and resists searching for other, potentially better, solutions. Design fixation is the antithesis of design exploration; it is design lack-of-exploration. Designers sometimes

get fixated on ideas generated early in the design process and tenaciously “cling to major design ideas and themes in the face of what at times might seem insurmountable odds,” (Rowe, 1987, p. 32) maybe due to the great cost associated with synthesizing a solution from the vast amount of information related to a design problem. Other reasons might be that design tools are based on theoretical models that do not correlate with designer’s cognitive processes (Smithers, 1994), or that they are too cumbersome to provide real-time interactive exploration at speeds that parallel the rate at which designers can think of new ideas.

2.3.1. Space planning and design thinking

Akin et al. (1988) created a computer program to create architectural space layouts called ‘HeGeL’, which stands for ‘Heuristic Generation of Layouts.’ This work could have been reviewed in the previous section on computer-aided space planning. However, their purpose was not so much to implement another method of automated space planning, but to validate a paradigm they developed for how designers work and possibly think. They developed this paradigm from observations of designers in action. In their words, “our interest in this research lies in understanding and modeling the design process as a cognitive skill,” and “we describe a system that simulates the behavior of designers as recorded in our protocol experiments” (Akin et al., 1988, p. 414).

The elements of their approach consist of design units (spaces), predicates (design objectives), and a constructive placement methodology using a generate-and-test search strategy. Similar to the way designers might approach a manual design problem, a number of predicates are selected for consideration and are active while all others are passive, then design units are placed in the potential building area using these predicates. If the placement is successful a next set of predicates are chosen,

otherwise the previous predicates are ‘backtracked’ and another set chosen. This process continues until all design units are successfully placed, or if the problem is over constrained until all possible search paths have ended in failure.

Their fundamental assertion is that “a designer has a vast amount of knowledge that is incrementally brought to bear on spatial design problems” (Akin et al., 1988, p. 430). While this is true, and although they do not propose this method as a way to automate space planning in practice, it may not be appropriate to model digital design tools completely on *observed* practices. The way designers verbally articulate their thought processes, in this case as an incremental process, might be a result of the limitation of currently available tools. New digital design tools that remove the requirement to think incrementally about parts of a problem might reveal new characteristics of design cognition.

2.3.2. *Alexander’s ‘Notes on the Synthesis of Form’*

As noted in the introduction, in ‘Notes on the Synthesis of Form,’ Alexander (1964) describes the space planning process using a wide variety of physical terms such as ‘force,’ ‘viscosity,’ ‘effortless contact,’ ‘frictionless coexistence,’ ‘equilibrium,’ ‘stress,’ ‘strength of interaction,’ and ‘physical influence.’ He uses this terminology throughout much of his discussion about the design process, and is an important reference for the work presented here because this terminology is so prevalent. But ironically, the essential idea proposed by Alexander has little or nothing to do with physics or physicality.

The process Alexander describes is one of a mathematical decomposition of a list of design variables into sets of related variables. A simplified version of the entire process is as follows:

1. List and define individual variables, which defines a potentially large set of ‘misfit’ variables called M .
2. Define the relationships between all the variables, which defines a set of links called L .
3. Analyze the graph of these links, called $G(M, L)$, to arrive at a decomposition of the variables into a hierarchy of smaller sets.
4. For each leaf node in G , draw a diagram that represents the essence of the set of variables contained in that node.
5. For each node containing sub-nodes, combine the smaller diagrams into a larger diagram.
6. Continue up the graph hierarchy to the root node, combining multiple diagrams into larger diagrams. The root diagram then contains the basis for a design that can be further refined.

Alexander uses the idea of forces of design to create diagrams of parts of a design, with the intent that the diagrams aid the design process by helping to understand how the parts interact with each other. His original intent was to articulate a mathematical approach to creating these diagrams. But subsequent research and experience showed him that it was not necessary to use a complicated mathematical process but that those diagrams could easily be created experientially. This thought process evolved into the excellent and well known series of books based on *The Timeless Way of Building* (Alexander, 1979) and *A Pattern Language* (Alexander et al., 1977).

The importance of this work is not so much in the mathematical process described, which Alexander admits is flawed, but in the description of the design pro-

cess itself. Describing the design process in so many physical terms sheds light on some important characteristics of that process, either on the design process itself, or on the designer's mental processes. If that description has any validity, and the continued extensive referencing of this work despite the questionable nature of the underlying proposal suggests that it is valid, then perhaps a methodology that is based on fundamentally similar characteristics should be explored. The approach presented in this dissertation can then be thought of as revisiting the supposition that a complicated mathematical process is needed to create diagrams, and seeing if it is possible to use this 'complicated' mathematical means to achieve a useful design process while hiding the complication in the underlying physical simulation.

2.4. Physically based methods in space planning

The most important reference related to this work is the Ph.D. dissertation of Mikako Harada (Harada, 1997), part of which is also found in Harada et al. (1995). She also uses a physically based approach to manipulate architectural floor plans, as well as volumetric massing and similar floor planning problems such as circuit board layout and page layout. The motivation for her work was to provide a means to explore design spaces related to shape grammars. Early research in shape grammars typically focused on the definition of the shape grammar itself and on the generation of a usually large number of alternate designs, but was extremely difficult for non-experts to interact with. She used the physically based paradigm to provide a natural means with which to interact with floor plans. Physically based modeling typically requires that a model be transformed from one state to the next in a continuous manner, and discrete changes to the model, such as changing the topological relationship of two rooms by moving one room from one side to the other, cannot be done.

This limits the user from interactively engaging with the design by leaping from one discrete design state to another. Harada's major contribution was in defining and implementing a way to make these kinds of discrete changes to a model. Her method involves reading the direction of motion of the user's input device, determining when that motion would be limited by one or more dynamic constraints, then performing a local search of the available alternatives around that design state and performing the discrete transformation of the best local alternative.

Harada lists three areas of future work: "1) applying the technique to more general shapes, 2) experimenting with semantics of constraints, in particular with the emphasis on subjective design criteria, and 3) applying the technique to a broader class of problem domains." (Harada, 1997, p. 120) As will be shown in section 9.2, areas 1 and 3 are also areas of future work related to my research. My research to a certain extent addresses some of the issues in area 2. As will be discussed later, a constraint is a characteristic that *must* be met, while what I call a *design objective* is similar to a constraint in that it is a characteristic about a design that a designer desires to be met. A design objective can then be thought of as a fuzzy constraint, and the ones I describe in effect expand the semantics of constraints (design objectives) to the architectural domain. The mechanical metaphor used to describe, define, implement, and interact with these design objectives also provide an additional level of semantics. Harada notes in the discussion of area 2 that "ultimately we want to have a tool that allows designers to '*simulate*' their minds or to test, learn and evaluate their ideas more freely at the level of generating ideas. (emphasis added)" (Harada, 1997, p. 121) This is an extremely interesting statement in light of the cognitive theory of design that emerged from my work, which is discussed in section 8, "Theoretical Implications." The physically based space planning approach started out to be a potentially interesting means with which to automatically produce space plans,

but might in fact be a simulation of the cognitive processes active in a designer's mind during the act of design.

2.5. Why use physics in digital design?

There are many established theories on design processes, but most, if not all, are static, in that they are based on an analysis of the process and products of design as static images. Those that might be dynamic in some way are dynamic in the sense of movement or change over time. For example, when discussing dynamic visualization in architecture Koutamanis says:

Dynamic visualization is often presented as the pinnacle of architectural representation, the fullest form of visual realism. By including *movement* of one sort or another in a three-dimensional representation, the designer adds depth and time to the subject under controlled conditions, i.e., in the framework of a specific event or state. As a dynamic description is a sequence of *static*, normally photorealistic images, the results can be superior to other representations for visual inspection, analysis and communication. (emphasis added)

He goes on to discuss dynamic visualization strictly in terms of cinematographic filming techniques.

Despite this historical focus on the static nature of design products, others are beginning to recognize the dynamic possibilities. Mitchell and McCullough (1995) titles the conclusion of a chapter on Animation “Unfreezing Images”, and says “The real world moves and changes, but designers have worked for centuries with frozen images - static structures of lines embedded in paper fibers. Now those images can be animated - brought to life - as, according to legend, was Pygmalion.” (Mitchell

and McCullough, 1995, p. 312)

But rarely is the term dynamic meant in the context of ‘dynamical’ in the sense of objects reacting to each other according to established laws. After learning the techniques of physically based modeling, I thought it would be useful to think about how those techniques could be applied to general design processes by applying them to the specific problem of space planning.

The question remains, though. Why use physics in digital design?

Here are three possible reasons:

1. The design process is traditionally physical and tactile.
2. Recent research in cognitive processes suggests that cognitive processes themselves, such as the design process, may be dynamical in nature.
3. Previous research supports this approach.

2.5.1. Design is traditionally physical

Traditional design processes are inherently physical in nature. McCullough notes that one of the key factors in the practicing of a craft is in touching the material being crafted, and that “. . . any fool can tell you that a craftsman needs to touch his or her work. This touch can be indirect . . . but it must be physical and continual, and it must provide control of whole processes.” (McCullough, 1996, p. x) The classic example is sculpture, where the artist has an intimate knowledge of the material being sculpted and a deep understanding of how tools affect that material. But this physical quality also extends to other non-sculptural arts. Musicians know the feel of the instrument. Painters know the feel of the canvas, paints, and brushes. Before computer aided design, architects knew the feel of paper, mylar, and pencil. Choosing the right pencil hardness; sharpening the pencil to a nice but not too sharp

point; using T-squares and triangles to draw lines of just the right thickness on the paper. All these skills promote an intensely physical connection between architects and their tools, and were skills required to produce quality architectural drawings. The artist knows how the tool is made, how it works, and how to manipulate it to achieve desired ends.

This physical connection between the artist and the tool is lacking in today's digital design applications. There is no longer that physical connection between the hand and the tool, and between the tool and the design. The artist has no deep understanding of why a tool acts like it does, and interfaces are rarely intuitive and easy to understand. So the artist is forced to expend mental energy focused on the tool rather than on the design. In Martin Heidegger's terms (Heidegger, 1962), traditional tools are 'ready-at-hand' unless they are broken, when they become 'present-at-hand,' while digital tools all too often feel 'present-at-hand' without providing the 'ready-at-hand' experience required to support the creative process. Lawson seems to put this concept in much simpler terms when he says "we probably work best when we think least about our technique" (Lawson, 1997, p. 11).

Since traditional design processes are physical in nature, the basis for that nature is physics, and since today's digital design tools lack a physical quality, it might be useful to incorporate a semblance of physics into their implementation. McCullough gives us hope that computers can eventually be used to restore a sense of touch to our practice when he writes "...our nascent digital practices seem more akin to traditional handicrafts, where a master *continuously coaxes a material*. This new work is increasingly continuous, visual, and productive of singular form; yet it has no material." (emphasis added) (McCullough, 1996)

2.5.2. *Design cognition may be dynamical*

Recent research in cognitive science suggests that the cognitive processes of the brain are better modeled as dynamical systems, rather than as digital computers. This new view is called the *dynamical hypothesis*, and extends the more traditional view, which is called the *computational hypothesis* (van Gelder, 1998).

Whereas the computational hypothesis models cognition as stored bits of information with algorithmic processes that modify their values and result in actions in the environment, in the dynamical hypothesis cognitive elements are directly influenced by each other, much as the Sun affects the motion of the Earth, which in turn affects the motion of the Sun.

Van Gelder illustrates the difference between the two concepts by describing how each would operate on a governor for a steam engine, which attempts to maintain a constant output for the steam engine; if the engine slows down the governor opens the throttle to increase its speed, and vice versa. He summarizes the differences between a computational governor and a dynamical governor as follows:

Instead of cycles of inputs, symbolic representations, rule-governed, atemporal computations, and outputs, we have the continual mutual influencing of two quantities. This influencing is very subtle (though mathematically describable): the state of one quantity is continually determining how the other is accelerating and vice versa. This relationship is very unlike the relationship between a digital symbol and its referent. (van Gelder, 1999, p. 5)

I discuss the contrasts between the dynamical and computational hypotheses more fully in section 8.3, as one argument for *dynamical design imagery*, the new cognitive theory of design that emerged as a result of this work.

See also Port and van Gelder (1995), Dietrich and Markman (2000), and Kelso (1995) for recent ideas on the emerging role of dynamics in cognitive science.

If the cognitive processes of the brain can be shown to be part of a dynamical system, then a specific cognitive process is also likely to be a part of a dynamical system. The act of design by a human is a specific cognitive process, which can then be analyzed in relation to the dynamic hypothesis. If the dynamical hypothesis proves to be valid, then a greater understanding of the design process as a dynamic system, and the application of that understanding to the design of tools that aid in the design process should yield better design tools.

2.5.3. Previous attempts support applying physics to design processes

As noted earlier in this section, several key pieces of research either support or are directly related to the application of physics to design processes. Alexander describes the design process using many physical terms. Qin and others use physically based methods for direct manipulation of geometric models, specifically free-form surfaces. Most important, Harada uses physically based methods for direct manipulation of space plans to support design exploration of shape grammars.

These three arguments suggest that designers need to begin to think of their designs in dynamic, fluid, life like terms. One way to enable designs to come alive is to make them respond to the objectives of the designer the way real objects respond to forces acting upon them in the real environment. Just as objects respond to physical forces, so can design objects respond to design forces. Physically based modeling is the means for simulating real object behavior, so it may be useful to employ physically based techniques in the design process.

Before presenting how physically based techniques can be applied as mechanical

metaphors of the elements in an architectural space plan, we need to review the basics of those techniques.

3. REVIEW OF PHYSICALLY BASED MODELING

This section provides a background on physically based modeling principles, since they form one of the central themes of this study and have rarely been applied to space planning problems. A basic understanding of these principles is necessary to understand how they are applied later to the space planning problem.

3.1. Definitions

The following definitions of terms in physically based modeling will be useful in the ensuing discussion:

3.1.1. Newton's first law of motion

Newton's first law of motion is "if there is no net force acting on a body, it will continue in its state of rest, or will continue moving along a straight line with uniform velocity." (Williams et al., 1976)

3.1.2. Newton's second law of motion

Newton's second law of motion is "the acceleration of a body is directly proportional to the net force exerted on the body, is inversely proportional to the mass of the body, and is in the same direction as the force." (Williams et al., 1976)

3.1.3. Newton's third law of motion

Newton's third law of motion is "whenever one body exerts a force on another, the second body exerts on the first a force of equal magnitude in the opposite direction." (Williams et al., 1976)

3.1.4. *Damp*

Damp is a transitive verb meaning “to check the vibration or oscillation of (as a spring or voltage).” (Merriam-Webster, 1995)

3.1.5. *Dampen*

Dampen is a verb meaning “to check or diminish the activity or vigor of.” It also is an intransitive verb meaning “to become deadened or depressed.” (Merriam-Webster, 1995)

3.1.6. *Dashpot*

A *Dashpot* is “a device for cushioning or damping a movement (as of a mechanical part) to avoid shock.” (Merriam-Webster, 1995)

3.1.7. *Viscosity*

Viscosity is “the property of resistance to flow in a fluid or semifluid.” (Merriam-Webster, 1995)

3.1.8. *Point*

A *point* specifies a position by defining three scalar values each representing a distance from the origin along a different coordinate axis (for example, (1, -2, 3) along the (x, y, z) axes). A point will be represented in a formula by a term with a line over it, such as \bar{p} .

3.1.9. *Vector*

A *vector* specifies a direction and a distance. A vector is uniquely defined by the difference between two points. Thus a single point and (implicitly) the origin define the vector whose direction is the direction from the origin to the point, and whose length is the distance from the origin to the point. A vector will be represented in a formula by a bold term, such as \mathbf{v} .

3.1.10. *Kinetic energy*

Kinetic energy is the energy of a mass associated with its motion.

3.1.11. *Potential energy*

Potential energy is the energy of a mass relative to its displacement from another position.

3.1.12. *Equilibrium*

Equilibrium is “the state of a body in which there are no unbalanced forces or torques acting on it.” (Williams et al., 1976)

3.2. **Example: Thrown ball**

A classic example of a physical system that can be simulated is that of a thrown ball with fixed mass. At the point in time when the ball leaves the thrower’s hand it has a position, and a velocity (defined as the rate of change of position in a specific direction). If this ball were to be thrown in the absence of gravity or air friction, as in space, it would continue moving with the same velocity until it came under the influence of new forces. If it were thrown on earth, it comes under the influence of

a number of forces, such as the force of gravity or the force of the air's drag on the surface of the ball. In a real system, each of these forces acts continuously on the ball; that is, they act at all instants in time. The task of a physically based simulation is to approximate the actual trajectory of the ball while accounting for all known forces on it. This simulation is accomplished through the process of numerical integration, a process of computing integrals at discrete rather than continuous points in time. The key point to understand in numerical integration is that it provides only an *estimation* of the integral. A primary focus in physically based simulation is to develop and use more accurate numerical integration methods.

Figure 3-1 shows a ball with mass m , position $\bar{p}(t)$, and velocity $\mathbf{v}(t)$ at time t . The forces acting on the ball include gravity \mathbf{f}_g , drag $\mathbf{f}_d(t)$, and wind $\mathbf{f}_w(t)$. The solid curve represents the actual path the ball may take under the influence of these forces. The dotted curve represents the estimated path the ball may take as a result of using numerical integration in a simulation.

3.3. Elements of a physical system

Within the context of this work, a physical system consists of a set of matter-based movable objects, a set of forces that may act upon those objects, and a means of simulating the system in a computational environment. Newtonian physics will suffice, as it approximates the everyday experience of humans; black holes, string theory, quantum mechanics, and relativity will be ignored. Additional concepts include collision detection (determining when two objects intersect each other), dynamic constraints (maintaining specified constraints in a dynamic system, such as a roller coaster constrained to roll on its tracks), and user interaction (how users can interact with the elements of the system).

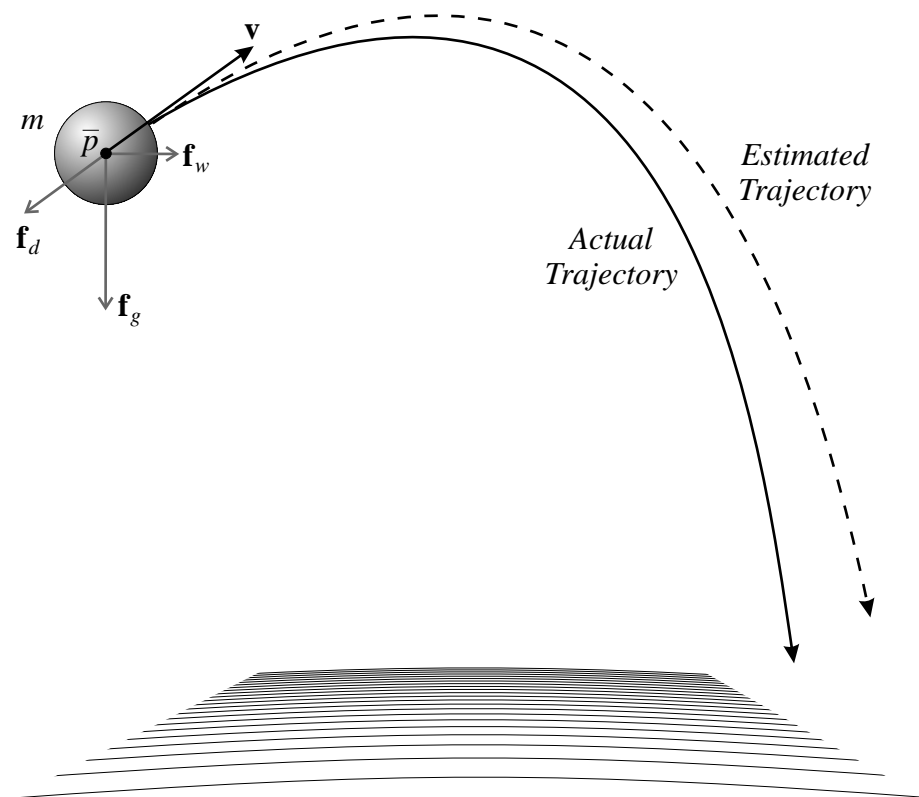


Fig. 3-1. A ball thrown through the air

3.3.1. Physical object

A *physical object* is an object that has matter and that can be acted upon to change its position. A state specifying its position and velocity can be defined for it at a single point in time.

3.3.1.1. Mass

A physical object has a *mass* (m), a measure of the matter existing in the object. A mass typically consists of a positive scalar value in some predefined units (such as 1.2 grams).

3.3.1.2. *Position*

A physical object has a *position* (\bar{p}), a measure of its location within a specified coordinate system. A typical coordinate system is the Cartesian coordinate system, where a position is represented by a point: three scalar values each representing a value along a different orthonormal coordinate axis (for example, (1, -2, 3) along the (x, y, z) axes).

3.3.1.3. *Velocity*

A physical object has a *velocity* (\mathbf{v}), or speed in a specific direction in a specific coordinate system, which is a measure of its rate of change of position. In the Cartesian Coordinate system, velocity is represented by a vector, three scalar values each representing a speed along a different coordinate axis (for example, (1, -2, 3) along the (x, y, z) axes). If the velocity is thought of as a point, the direction component of the velocity is the direction from the origin to its position, while the speed component is the length of the vector from the origin to its position.

3.3.1.4. *State*

The *state* of a physical object is the set of values that fully describe its current dynamic condition. For an object without rotational inertia, this is its position and velocity. The state of an object is changed by the application of external forces. The state of such an object with fixed mass at some time t is captured by its position $\bar{p}(t)$ and its velocity $\mathbf{v}(t)$.

3.3.2. Basic motion

The velocity of an object at a specific point in time is a measure of how its position is changing at that time,

$$\mathbf{v}(t) = \dot{\bar{\mathbf{p}}}(t).$$

For example, given a coordinate system measured in feet and time measured in seconds, an object at time $t_0 = 0$ might be at position $[1, 2, 3]$, with a velocity of $[1, 1, 0]$ feet per second. Assuming no forces are acting on the object, the position of the object at some later time is

$$\bar{\mathbf{p}}(t_0 + \Delta t) = \bar{\mathbf{p}}(t_0) + \Delta t \mathbf{v}(t_0).$$

If $\Delta t = 2$ then

$$\bar{\mathbf{p}}(t_0 + 2) = [1, 2, 3] + 2[1, 1, 0] = [3, 4, 3].$$

The acceleration of an object at a specific point in time is a measure of how its velocity is changing at a specific time,

$$\mathbf{a}(t) = \dot{\mathbf{v}}(t) = \ddot{\bar{\mathbf{p}}}(t).$$

Loosely using the same example as above, but in this case with some force producing an acceleration of $[0, 1, 0]$ feet per second squared, the velocity of the object at some later time is

$$\mathbf{v}(t_0 + \Delta t) = \mathbf{v}(t_0) + \Delta t \mathbf{a}(t_0).$$

If $\Delta t = 2$ as before then

$$\mathbf{v}(t_0 + 2) = [1, 1, 0] + 2[0, 1, 0] = [1, 3, 0].$$

The problem with the examples is that they only work for uniform velocities

or accelerations. In realistic systems with varying accelerations, and velocities that continuously change, the above calculations quickly produce errors in succeeding states and ultimately unrealistic motion. Very small time steps can improve the accuracy somewhat, but result in greatly increased computation time. Despite these problems, the examples do give a basic sense of the relationships between position, velocity, and acceleration.

3.3.3. Force and acceleration

A *force* acts upon physical objects in the classical method of Newtonian dynamics. The basic equation of force is Newton's Second Law of Motion, which can be written $\mathbf{f} = m\mathbf{a}$. Acceleration is a measure of the rate of change of velocity of an object in a specific coordinate system. The acceleration of a physical object is proportional to the force that is applied to it. The constant of proportionality is known as the mass of the object.

Forces are categorized here based on the number of objects they act upon in a simulation and the manner of their application. This categorization is not absolute or even physically correct (that is, a physicist would probably object to it), but is defined for a class of physical systems used in computer animation. Forces are specified in this section with the term f_i , where i designates the type of force being described.

3.3.3.1. Unary forces

Unary forces act on a single object independent of other objects. Technically speaking, unary forces do not exist in classical physics. They are used as a means to simplify parts of a physical simulation.

Gravity. *Gravity* is a force resulting from the interaction between two objects dependent on their masses and the distance between them. Newton's law of universal gravitation can be written

$$\mathbf{f}_{gravity} = \mathbf{G} \frac{m_1 m_2}{d^2},$$

where m_1 and m_2 are the masses of the two objects, d is the distance between their centers of mass, and \mathbf{G} is the universal gravitational constant.

In some physical systems, such as objects on the surface of the Earth, the force of gravity does not change significantly throughout the range of possible locations in the system, so it can be treated as a constant force applied to all objects in the system. One mass (the Earth) is so large compared to the other that its acceleration can be ignored.

The force due to gravity on an object can be computed using

$$\mathbf{f}_{gravity} = m\mathbf{g},$$

where m is the mass of the object, and \mathbf{g} is the gravitational constant.

Viscous drag. *Viscous Drag* is actually a force on a solid object resulting from its motion through a 'fluid' medium (a liquid or a gas). It has the effect of slowing down the object, and is usually considered to be directly proportional to the velocity of the object. The object and the fluid interact with each other and apply forces to each other.

The force due to viscous drag on an object can be computed using

$$\mathbf{f}_{drag} = -k_{drag}\mathbf{v}$$

where k_{drag} is the drag coefficient, and \mathbf{v} is the velocity of the object.

3.3.3.2. *n*-ary forces

n-ary forces are applied to a fixed set of objects. Typical are forces that are applied between two objects.

Spring. A *spring* connects two objects and applies a force to each proportional to the distance between them. A spring has a rest length, at which it applies no forces. If the spring is longer than the rest length, it applies forces pulling the objects together, and if it is shorter than the rest length, it applies forces pushing them apart.

The forces due to a spring on two objects at positions \bar{a} and \bar{b} can be computed using

$$\mathbf{f}_{spring}^a = -k_s(|\Delta\mathbf{x}| - r) \frac{\Delta\mathbf{x}}{|\Delta\mathbf{x}|}$$

$$\mathbf{f}_{spring}^b = -\mathbf{f}_{spring}^a$$

where k_s is the spring coefficient, $\Delta\mathbf{x}$ is the direction vector between the two objects ($\Delta\mathbf{x} = \bar{a} - \bar{b}$), $|\Delta\mathbf{x}|$ is the distance between them, and r is the rest length of the spring.

Damper. A *damper* applies a force to two objects proportional to the difference in their relative velocities. It works much like a screen door closer, trying to keep the door from slamming shut. If the objects are moving very fast toward or away from each other, the damper applies a large force to each trying to slow them down (relative to each other). If they are not moving at all towards or away from each other, the damper applies no forces. The speed of the two objects as a system has no effect on the damping force, nor does motion tangential to the direction between the objects.

The force due to a damper on two objects at positions \bar{a} and \bar{b} , with velocities

\mathbf{v}_a and \mathbf{v}_b , respectively, can be computed using

$$\mathbf{f}_{damping}^a = -k_d \left(\Delta \mathbf{v} \cdot \frac{\Delta \mathbf{x}}{|\Delta \mathbf{x}|} \right) \frac{\Delta \mathbf{x}}{|\Delta \mathbf{x}|}$$

$$\mathbf{f}_{damping}^b = -\mathbf{f}_{damping}^a,$$

where \cdot is the dot product between two vectors, $\Delta \mathbf{x}$ is defined as above, k_d is the damping coefficient, and $\Delta \mathbf{v}$ is the difference between the velocities of the objects ($\Delta \mathbf{v} = \mathbf{v}_a - \mathbf{v}_b$). The middle term ($(\Delta \mathbf{v} \cdot \frac{\Delta \mathbf{x}}{|\Delta \mathbf{x}|})$) is the relative velocity between the objects; that is, it is the component of $\Delta \mathbf{v}$ parallel to the vector between the two objects, ignoring the tangential component.

3.3.4. Example: Mass-spring-damper system

Figure 3-2 puts many of the elements just discussed together. It shows a simple mass-spring-damper system consisting of two masses m_0 and m_1 at positions \bar{p}_0 and \bar{p}_1 , connected by a spring with spring constant k_{01} and a dashpot with damping constant d_{01} , and predefined with a desired rest length r_{01} . The current length l_{01} of the spring is the magnitude of the vector between the positions \bar{p}_0 and \bar{p}_1 at the current time. The spring exerts forces with magnitude proportional (with proportionality constant k_{01}) to $l_{01} - r_{01}$. The direction of these forces will be along the line connecting the point masses. As the masses move farther from each other, the spring forces try to move them closer, and as they move closer these forces try to separate them. The dashpot, attached in parallel with the spring, damps the motion of the masses by producing forces proportional (with proportionality constant d_{01}) to their relative velocity towards or away from each other, thus reducing the kinetic energy introduced by the spring forces. The net result of the spring and damper forces is shown as \mathbf{f}_0 and \mathbf{f}_1 in the figure (note that $\mathbf{f}_0 = -\mathbf{f}_1$).

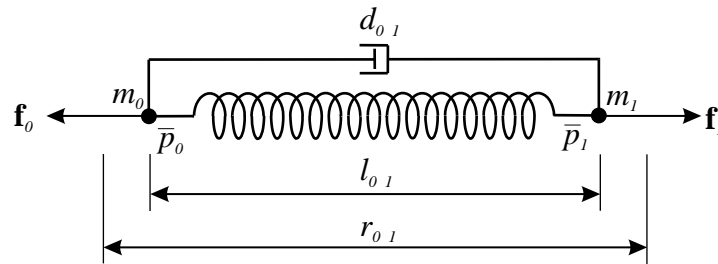


Fig. 3-2. A mass-spring-damper system

3.3.5. Numerical simulation

Given a physical system containing a set of physical objects and their states, and a set of forces, *numerical integration* is used to compute the next system state. A numerical simulation is a means of computationally estimating how the state of a physical system changes; that is, given a state at time t , numerical simulation computes the next state at time $t + \Delta t$, where Δt is the change in time or the *time step*.

As mentioned before, a real physical system changes *continuously* in time, while numerical simulation estimates the state at *discrete* points in time. The accuracy of the simulation, and thus the quality of it, depends on the design of the numerical simulation. The challenge in designing a numerical simulation is to find the right balance of time step and numerical integration method that produces a quality simulation in a reasonable amount of computational time.

3.3.5.1. Time step

A *time step* is a discrete time interval over which the next state will be computed. For example, given a time scale measured in seconds, the time step might be set to 1/10 seconds. In general, the smaller the time step, the more computational

resources required to compute a specified time range, and the better the accuracy of the simulation.

3.3.5.2. Numerical integration

Each state of a dynamic simulation is computed using numerical integration, which in the context of physically based simulation is a process of estimating a future state based on a set of current conditions. Two interrelated concerns arise when making estimates using numerical integration, accuracy and stability. Both depend on the step size and the quality of the numerical integration method, and are greatly affected by the dynamics of the system being simulated.

There are two basic types of integration methods discussed here, non-adaptive and adaptive. In both approaches, a step size is specified at the beginning of the simulation. In non-adaptive methods, the step size does not vary during the simulation, whereas in adaptive methods, it can vary throughout the simulation as it attempts to adapt to estimates of error in the simulation. Much of the following discussion can be found in Witkin and Baraff (1997).

Non-adaptive methods. With non-adaptive integration methods, the state at the end of a specified uniform time step is computed. That computation will always contain some error, but the time step does not change based on the magnitude of that error. In certain physical systems this error can lead to noticeably unrealistic physical behavior. The Euler and Runge-Kutta Methods are examples of non-adaptive integration methods.

Euler's method. The simplest numerical integration method and one that helps in understanding the other methods is Euler's method. It uses the method described in section 3.3.2 by taking a step in the direction of the current velocity. Euler's

Method is

$$\mathbf{x}(t+h) = \mathbf{x}(t) + h\dot{\mathbf{x}}(t),$$

where Δt is now replaced by h , and $\mathbf{x}(t)$ is the state vector at time (t) containing position and velocity for every object in the system, $\mathbf{x}(t) = [\bar{p}_0, \mathbf{v}_0, \bar{p}_1, \mathbf{v}_1, \dots, \bar{p}_{n-1}, \mathbf{v}_{n-1}]$ where n is the number of objects in the system. The derivative of the state vector is $\dot{\mathbf{x}}(t) = [\bar{v}_0, \mathbf{a}_0, \bar{v}_1, \mathbf{a}_1, \dots, \bar{v}_{n-1}, \mathbf{a}_{n-1}]$. Since the accelerations are all functions of state, if we know the dynamics of a system we can always find a function $f(\mathbf{x}, t)$ such that $\dot{\mathbf{x}} = f(\mathbf{x}, t)$.

Euler's method is very simple, and very quick to solve for a single time step, but it produces large errors very quickly and is highly unstable, as shown in figure 3-3. 3-3a shows what is supposed to be circular motion, but no matter how small the time step is set, the estimated path will always spiral outward. 3-3b shows what is supposed to be a converging path, but if the time step is too large, the estimated path will diverge.

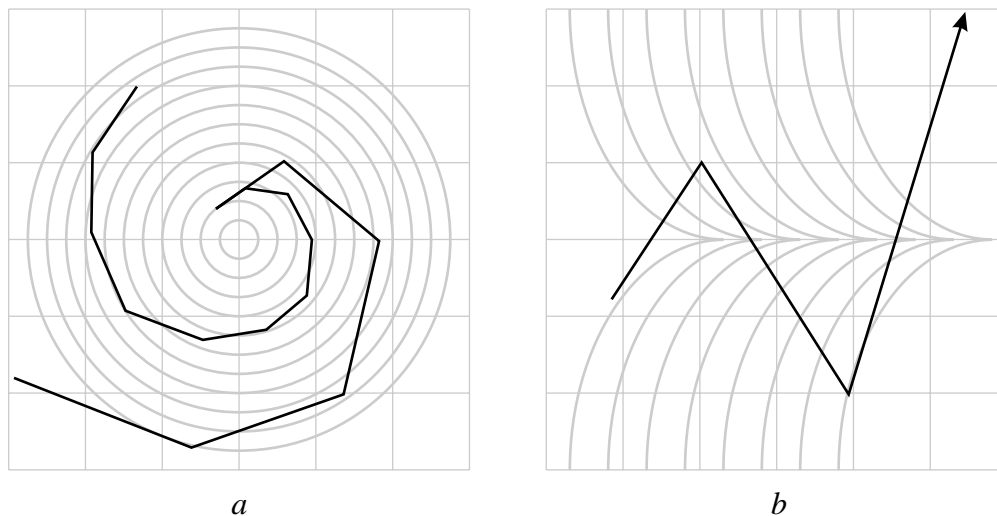


Fig. 3-3. Problems with Euler Integration [From Witkin and Baraff (1997, p. B4)]

Taylor series. Given a state \mathbf{x} at time t_0 and a time step h , we can express the new state $\mathbf{x}(t_0 + h)$ as the sum of the initial state and an infinite number of its derivatives. This is done using the *Taylor series*,

$$\mathbf{x}(t_0 + h) = \mathbf{x}(t_0) + h\dot{\mathbf{x}}(t_0) + \frac{h^2}{2!}\ddot{\mathbf{x}}(t_0) + \frac{h^3}{3!}\mathbf{x}^{iii}(t_0) + \dots + \frac{h^n}{n!}\frac{\partial^n \mathbf{x}}{\partial t^n} + \dots$$

If we know $\mathbf{x}(t_0)$ and all of its derivatives then we can use the Taylor series to compute $\mathbf{x}(t_0 + h)$ exactly, as long as there are no singularities between t_0 and $t_0 + h$ (a singularity is a point at which one or more derivatives of \mathbf{x} are undefined or unbounded, which, for example, can occur at a collision).

Euler's Method is simply a truncation of the Taylor series; it throws away everything higher than the first order derivative $\dot{\mathbf{x}}(t_0)$, which we know because it is part of the state or can be calculated using Newton's second law ($\mathbf{x}(t) = (\bar{p}(t), \mathbf{v}(t))$, $\dot{\mathbf{x}}(t) = (\dot{\bar{p}}(t), \dot{\mathbf{v}}(t)) = (\mathbf{v}(t), \mathbf{a}(t)) = (\mathbf{v}(t), \frac{\mathbf{f}}{m}(t))$). The measurement of the resulting error is the dominate term in those that are thrown away, which is Oh^2 (read as Order h^2) when h is smaller than one. Since we know none of the higher order derivatives, in order to get more accuracy in the computation we must find a method for estimating some of the higher order terms. One such class of methods is called *Runge-Kutta*, which takes a weighted average of a number of estimates of future states.

Midpoint method (2^{nd} order Runge-Kutta method). The midpoint method uses the Taylor series out to the second-order term ($\frac{h^2}{2!}\ddot{\mathbf{x}}(t_0)$), which it approximates, and has error $O(h^3)$ because it throws away the rest. It is shown graphically in figure 3-4. \bar{p}_t is the position of the object at the beginning of the time step. The curve that ends at point \bar{p}_{t+h} is the actual path of an object during a single time step. Point \bar{p}'_{t+h} is the result of taking an Euler step; that is, moving from \bar{p}_t in the direction of its velocity for one time step.

As noted earlier, the function $f(\mathbf{x}(t_0), t_0)$ computes the derivative of the state

vector, $\dot{\mathbf{x}}(t_0)$. The midpoint method takes an Euler step,

$$\Delta \mathbf{x} = hf(\mathbf{x}, t),$$

evaluates f at the midpoint of this step to get estimates of velocity and acceleration halfway through the time step,

$$f_{mid} = f\left(\mathbf{x} + \frac{\Delta \mathbf{x}}{2}, t + \frac{h}{2}\right),$$

and then takes a step in the direction of *that* velocity one time step,

$$\mathbf{x}(t+h) = \mathbf{x}(t) + hf_{mid},$$

to reach \bar{p}''_{t+h} . Putting this all together yields

$$\mathbf{x}(t+h) = \mathbf{x}(t) + h\left(f(\mathbf{x}(t)) + \frac{h}{2}f(\mathbf{x}(t), t), t + \frac{h}{2}\right), \quad (3.1)$$

or assuming that f does not directly depend on the time,

$$\mathbf{x}(t+h) = \mathbf{x}(t) + h\left(f(\mathbf{x}(t)) + \frac{h}{2}f(\mathbf{x}(t))\right).$$

See Witkin and Baraff (1997) for the derivation.

For identical time steps, the computational cost of the midpoint method is roughly twice that of Euler's method, but the midpoint method produces better results because it is more accurate. $f(\mathbf{x})$ dominates the computation time and since the midpoint method must compute $f(\mathbf{x})$ twice and Euler's method once, taking a midpoint step with a time step of h is similar in computational cost to taking two Euler steps with a time step of $\frac{h}{2}$. However, the midpoint method, with error $O(h^3)$, produces less error than Euler's method, with error $O(h^2)$, so produces better results. In practice, most dynamics simulations can be run faster using the midpoint method than using Euler's method while maintaining the same level of accuracy.

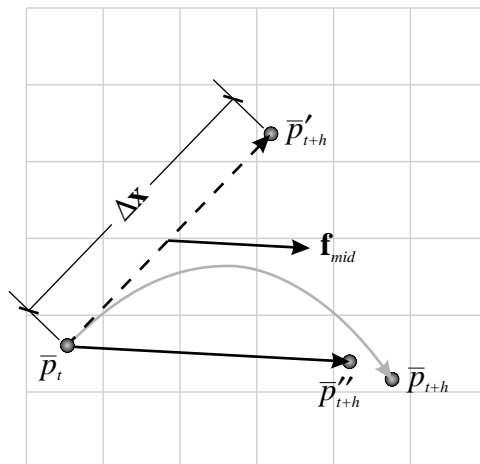


Fig. 3-4. Midpoint method [From Witkin and Baraff (1997, p. B6)]

In addition, Euler's method cannot be used to simulate certain kinds of motion no matter how small the time step. For example, when simulating circular motion, refer back to figure 3-3a, Euler's method will always result in spiral motion but the midpoint method will produce circular motion.

4th order Runge-Kutta method. The midpoint method can also be called a 2nd order Runge-Kutta method, since it estimates the second order term in the Taylor series. The 4th order Runge-Kutta method, also called RK4, estimates the second through the fourth terms in the Taylor series, evaluates $f(\mathbf{x})$ four times, and has error $O(h^5)$. The following set of equations, similar in form to equation 3.1, is used

to compute $\mathbf{x}(t_0 + h)$ using the RK4 method:

$$\begin{aligned} \mathbf{k}_1 &= hf(\mathbf{x}_0, t_0) \\ \mathbf{k}_2 &= hf\left(\mathbf{x}_0 + \frac{k_1}{2}, t_0 + \frac{h}{2}\right) \\ \mathbf{k}_3 &= hf\left(\mathbf{x}_0 + \frac{k_2}{2}, t_0 + \frac{h}{2}\right) \\ \mathbf{k}_4 &= hf(\mathbf{x}_0 + k_3, t_0 + h) \\ \mathbf{x}(t_0 + h) &= \mathbf{x}_0 + \frac{1}{6}\mathbf{k}_1 + \frac{1}{3}\mathbf{k}_2 + \frac{1}{3}\mathbf{k}_3 + \frac{1}{6}\mathbf{k}_4. \end{aligned}$$

As with the midpoint method, taking a RK4 step with a time step of h is similar to taking two midpoint steps with a time step of $\frac{h}{2}$. However, the RK4 method, with error $O(h^5)$, produces less error than the midpoint method, with error $O(h^3)$, so produces better results, while remaining faster.

Adaptive methods. Adaptive integration methods adjust their result based on some measurement of the error produced during a time step. An acceptable error range is specified. If the error during a time step exceeds the higher threshold the time step is reduced, the state is recomputed, and the error measured again. This process continues until the error is within the acceptable range and the resulting time step is used for the next step. If the error during a time step is below the lower range the time step can be increased for the next step. Two common adaptive integration methods are Runge-Kutta-Felberg and Runge-Kutta-Cash-Karp. Both of these are fifth order integration methods where the fifth-order term is used as the measurement of the error against which the error threshold is compared. This measurement of error is not to be confused with the *order* or error, which for these methods is $O(h^6)$. See Press et al. (1992) for a more complete explanation.

It appears that much more computation is required with adaptive methods, but

Table 3-1. Comparison of truncated Taylor series used in integration methods

Taylor series
$\mathbf{x}(t_0 + h) = \mathbf{x}(t_0) + h\dot{\mathbf{x}}(t_0) + \frac{h^2}{2!}\ddot{\mathbf{x}}(t_0) + \frac{h^3}{3!}\mathbf{x}^{iii}(t_0) + \dots + \frac{h^n}{n!}\frac{\partial^n \mathbf{x}}{\partial t^n} + \dots$
Truncated Taylor series on which Euler's Method is based:
$\mathbf{x}(t_0 + h) = \mathbf{x}(t_0) + h\dot{\mathbf{x}}(t_0) + O(h^2)$
Truncated Taylor series on which the Midpoint Method is based:
$\mathbf{x}(t_0 + h) = \mathbf{x}(t_0) + h\dot{\mathbf{x}}(t_0) + \frac{h^2}{2!}\ddot{\mathbf{x}}(t_0) + O(h^3)$
Truncated Taylor series on which the Runge-Kutta Method is based:
$\mathbf{x}(t_0 + h) = \mathbf{x}(t_0) + h\dot{\mathbf{x}}(t_0) + \frac{h^2}{2!}\ddot{\mathbf{x}}(t_0) + \frac{h^3}{3!}\mathbf{x}^{iii}(t_0) + \frac{h^4}{4!}\mathbf{x}^{iv}(t_0) + O(h^5)$

because the result is more accurate, and because the extra computation only occurs during error prone states, usually when accelerations are high, a larger basic time step can be used resulting in less overall computational expense. However, the main advantage is that the simulation can be run with precise error control at every time step and so is the preferred method if precision is of concern.

3.3.5.3. Accuracy and Stability

As noted in section 3.3.5.2, two problems that arise when using numerical integration are inaccuracy and instability. The accuracy and stability of a numerical method depends on how small the time step is, and the highest order derivative this is estimated.

The accuracy of each method has already been discussed, and is the order of error ($O(h^n)$ where n is the lowest order derivative discarded from the Taylor series). Table 3-1 shows again the Taylor series and the truncated versions of the three methods previously discussed. Comparing each method shows the additional term(s) that must be estimated for each method, and what part of the Taylor series is discarded, which provides the measurement of the order of error.

Physical simulations are notorious for *blowing up* when instability occurs. The simulation appears to behave normally, then all of a sudden becomes erratic until the elements explode outward because their positions and velocities are vastly outside their ‘normal’ or stable range.

The notion of *stability* will be discussed as it relates to the mass-spring-damper system in figure 3-5, which shows an object with mass (mass coefficient) m connected to a wall with infinite mass by a parallel spring and damper. The spring has spring coefficient k , rest length r , and current length l due to an applied external force f_{ext} , and the damper has damping coefficient d . The object can only move along the x -axis.

Our task is to choose an appropriate time step h so that the numerical simulation remains stable. If h exceeds a certain threshold, the system will become unstable and blow up.

The stability of this system is a function of the mass, spring, and damper coefficients, and the key to understanding this is to understand the *time constant* T of the system, which is a measure of how long it will take motion to die out when all forces

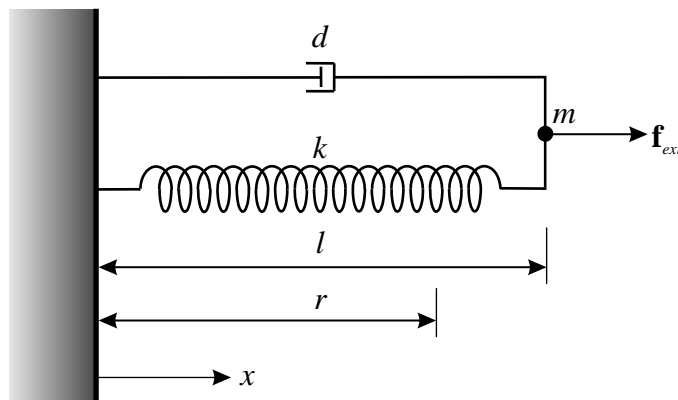


Fig. 3-5. Stability example: a 2^{nd} order linear system

are removed. Depending on the integration method, h needs to be some fraction of the minimum time constant (T_{min}) of the entire system. For example, using Euler's Method, h must be less than $2T_{min}$.

There are two kinds of motion that the object can have in this system, oscillatory and damped, and typically both. In oscillatory motion the object continually crosses back and forth over its resting position. In damped motion the object continually approaches its resting position without crossing over it. Table 3-2 attempts to simplify understanding the relationship between these three coefficients by essentially removing one of them from the system and taking the other two to extremes.

3.3.6. Collision detection and response

The discussion above describes the forces acting on an object and how those forces affect the object's motion, but it does not describe how objects interact with each other when they collide or attempt to occupy the same space at the same time. There are two collision tasks that need to be accomplished in a physical simulation, detecting when a collision has occurred between two objects, and then responding to the collision when it occurs (Moore and Wilhelms, 1988).

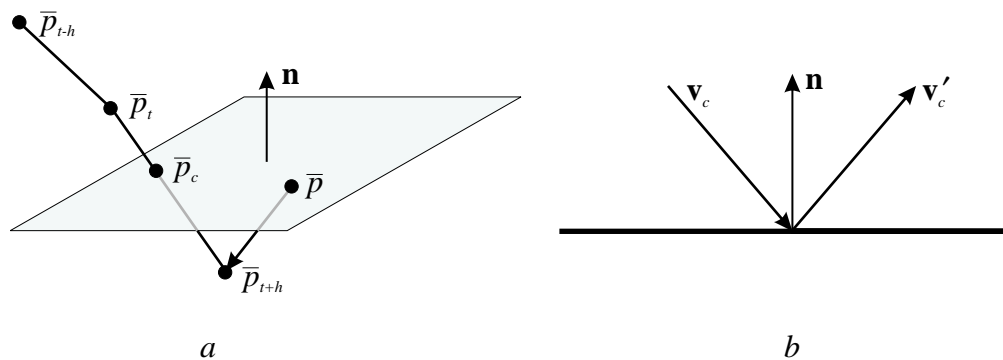


Fig. 3-6. Collision detection and response

Table 3-2. Stability depends on mass, spring, and damper constants

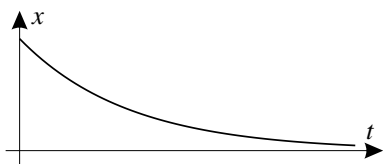

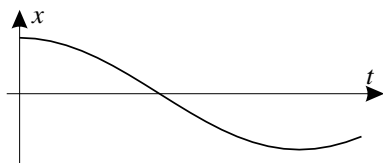
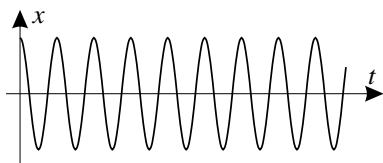
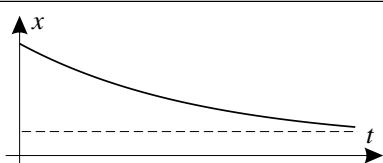
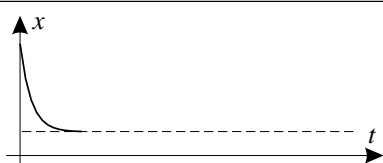
Tiny Mass	a: Strong Damper - Weak Spring	b: Weak Damper - Strong Spring
damped motion $l \neq r$	 <p>small spring forces large damping forces low velocities long time to rest ∴ large time step</p>	 <p>large spring forces small damping forces large velocities short time to rest ∴ small time step</p>
Dampless	c: Large Mass - Weak Spring	d: Small Mass - Strong Spring
oscillatory motion $l \neq r$	 <p>small spring forces large mass low velocities low frequencies ∴ large time step</p>	 <p>large spring forces small mass large velocities high frequencies ∴ small time step</p>
Springless	e: Large Mass - Weak Damper	f: Small Mass - Strong Damper
damped motion $v \neq 0$	 <p>large mass small damping forces long time to rest ∴ large time step</p>	 <p>small mass large damping forces short time to rest ∴ small time step</p>

Figure 3-6 shows a simplified method for detecting when a point object has collided with an arbitrary plane, and how to respond to that collision. In 3-6a, \bar{p}_{t-h} , \bar{p}_t , and \bar{p}_{t+h} are the object's positions at three points in time if there were no collision with the plane, which has normal \mathbf{n} pointing toward the non-collision side of the plane. The object crosses the plane between times t and $t + h$ at position \bar{p}_c . We determine a collision has occurred if $(\bar{p}_{t+h} - \bar{p}) \cdot \mathbf{n}$ is negative, where \bar{p} is an

arbitrary point on the plane. We can then determine the fraction s along the line \bar{p}_t and \bar{p}_{t+h} where \bar{p}_c occurs using

$$s = -\frac{d + \mathbf{n} \cdot \bar{p}_t}{\mathbf{n} \cdot (\bar{p}_{t+h} - \bar{p}_t)},$$

where d is found using the plane equation for $\mathbf{n} = [a, b, c]$

$$ax + by + cz + d = 0.$$

The time that the collision occurred can then be approximated as $t + sh$. This is only an approximation, because it assumes that there is uniform motion during the time step, when usually the velocity of the object is changing throughout the time step.

At this point in the simulation, the time is $t + h$. In order to respond to the collision we can back up the time to t , and compute the system's state at the approximate time the collision occurred $t + sh$. During a collision, what happens to an object's velocity is that it instantaneously (for our purposes) changes direction and magnitude. To respond to the collision then, we need to determine a new velocity vector. Figure 3-6b shows in two dimensions what happens to the velocity before and after the collision. \mathbf{v}_c is the velocity of the object prior to the collision, while \mathbf{v}'_c is its velocity after the collision, which can be found using

$$\mathbf{v}'_c = \mathbf{v}_c - 2(\mathbf{v}_c \cdot \mathbf{n})\mathbf{n},$$

(Whited, 1980). This equation assumes that there is no loss of motion during the collision. In a physical collision, however, some of the energy of motion is converted to other forms of energy, such as heat and noise. This loss of energy can be simulated by introducing an energy attenuation term, α , called the *coefficient of restitution*. α can range between 0 for total energy loss, to 1 for no energy loss. The equation then

becomes

$$\mathbf{v}'_c = \mathbf{v}_c - (1 + \alpha)(\mathbf{v}_c \cdot \mathbf{n})\mathbf{n}.$$

This discussion applies only to point objects colliding with immovable planes, and does not account for collisions between possibly rotating rigid bodies where the collision itself can introduce rotations within the colliding bodies. See Witkin and Baraff (1997) for an implementation of determining and responding to collisions between rigid bodies.

3.3.7. *Dynamic constraints*

Constraining the movement of objects in predetermined ways, such as simulating a roller coaster constrained to its track, introduces problems into a physically based simulation. One way to simulate this is to attach the simulated roller coaster to the track via a spring. As the roller coaster moves away from the track, the spring exerts a force on the roller coaster to move it back toward the track. One problem with this approach is that it leads to very loose or ‘goopy’ behavior, probably leading to an even sicker stomach. The stiffness of the spring can be increased, but this very soon results in a *stiff system*. As the stiffness of the system is increased, the time step needed to produce a satisfactory estimate of the next state must be decreased, and the computation time is considerably increased.

Dynamic constraints are often used in physically based simulations to meet this need. Instead of using springs, whenever a force would break a specified constraint, an additional *constraint force* is added to the system so that it does not. Constrained dynamics modifies the normal numerical simulation process by computing these constraint forces after normal forces are determined but before the next state is computed, as follows: 1) given a state, 2) compute the forces on the elements of the

system, 3) compute any additional forces necessary to maintain specified constraints, 4) then compute the next state. Figure 3-7a shows what is supposed to happen in the case of a physical bead on a circular wire. At time t , the bead is at position \bar{p}_t , its velocity \mathbf{v}_t will be tangent to the circle, and its acceleration \mathbf{a}_t will be such that at time $t + 1$ the bead's position will be at \bar{p}_{t+1} and its velocity \mathbf{v}_{t+1} will still be tangent to the circle. In an unconstrained numerical simulation, such as one using springs in figure 3-7b, it is possible for a force \mathbf{f}_t at time t to result in the bead's position at time $t + 1$ to be off the wire. Using dynamic constraints, an additional constraint force \mathbf{f}_c is added to \mathbf{f}_t to ensure that the bead stays on the wire, as shown in figure 3-7c.

See Witkin and Kass (1988) and Barzel and Barr (1988) for discussion on how dynamic constraints can be used in computer animation, and Witkin and Baraff (1997) for details on how to implement the mathematics needed to compute constraint forces.

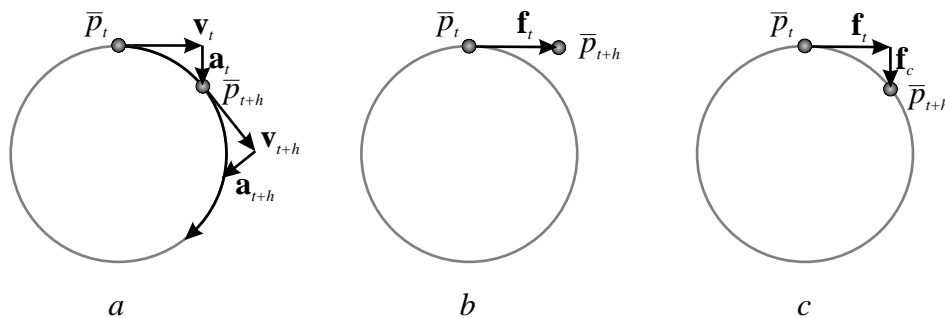


Fig. 3-7. Example of a dynamic constraint

3.3.8. *User interaction*

Interaction with the elements of a physically based model needs some special handling. In a modeling environment that is not physically based, such as traditional CAD, if the user wants to change the position of an object, they simply move the object. In a physically based simulation, moving an object using an ‘outside agent’ results in discontinuous, very large accelerations, which can introduce instabilities into the system. ((Witkin et al., 1990; Witkin and Welch, 1990), and (Witkin and Baraff, 1997, p. C9))

3.3.8.1. *Via force applicator*

One method of interacting with the elements in a physically based modeling system is to treat the user interactor as a force object in itself. Typically this is done with a spring with a zero rest length. When a user clicks on an object to move it, a spring is introduced into the simulation, with a rest length of zero, and both endpoints on the object to be moved. When the user moves the cursor, one end of the spring moves while the other remains connected to the object. The spring now introduces a valid force into the system that seeks to move the object toward the user’s cursor position.

3.3.8.2. *Via infinite mass*

Another method of interaction is to treat the object to be moved as if it temporarily has infinite mass. Any forces that get applied to the object then result in an acceleration of zero, since by Newton’s Second Law acceleration is given by $\mathbf{a} = \frac{\mathbf{f}}{m}$. Since this is the only place where the mass term is used, instead of storing the mass with each object, it is convenient to store its inverse, so the equation becomes $\mathbf{a} = m^{-1}\mathbf{f}$. When a user clicks on an object to move it, the object’s mass is temporarily set to

infinity by setting its mass inverse to zero. The object is moved around in the system in a seemingly traditional manner, but any interactions the object has with other objects are in terms of valid forces and accelerations.

3.3.9. Kinetic energy and dynamic equilibrium

The **kinetic energy** KE of a body is its energy of motion, and is dependent on its velocity v and mass m ,

$$KE = \frac{1}{2}mv^2.$$

For n bodies in a system, the total kinetic energy is

$$KE_{total} = \frac{1}{2} \sum_{i=1}^n m_i |\mathbf{v}_i|^2.$$

A system of springs, masses, and dampers is said to be in **dynamic equilibrium** when it has zero total kinetic energy. However, a system approaching zero total kinetic energy does so asymptotically. For practical purposes in a numerical simulation we define a threshold value KE_{min} , such that if $KE_{total} < KE_{min}$, it is considered to be in dynamic equilibrium.

3.4. Summary

Given the elements and concepts discussed in this section, the final step is to put them all together into an algorithm that performs the actual dynamics simulation. After setting up the initial conditions in the physical system being simulated, the basic algorithm continually loops through a process of computing any auxiliary variables that can be computed given a state and a time (for example, display characteristics such as a color based on velocity), displaying a single state, computing the forces on the objects, and using those forces in an integration method to estimate the next

```
Establish an initial state consisting:  
  A set of objects with mass, position, and velocity  
  A set of force types to be applied to the objects  
  An integration method  
  An initial time,  $t = 0$   
  A time step,  $h$   
  
Repeat:  
  Compute any auxiliary variables  
  Display the objects  
  Compute object accelerations from the state, time, and forces  
  Compute any forces needed to maintain dynamic constraints  
  Integrate to get the next state at time  $t + h$   
  Set  $t = t + h$   
Until  $t > \text{maximum time}$ 
```

Fig. 3-8. Basic numeric simulation algorithm

state. This process is outlined in figure 3-8.

Now that the basic elements and methods used in a physically based simulation have been presented, I can describe how they can be applied to design problems in general and space planning problems in particular.

4. A PHYSICALLY BASED APPROACH TO SPACE PLANNING

Physically based modeling will be used as the basis for modeling the dynamical behavior of cognitive design objects. A *design element* is an individual, configurable part of a design problem, and is represented as a mass with position and velocity. A design state is a specific configuration of a number of design elements at a discrete point in time. A *design objective* is an intended configuration of one or more design elements. According to Newton's second law of motion and assuming a constant mass, the only mechanism that can change an object's velocity, and consequently its position, is a force applied to it. Given this law and a mass representation of a design element, a design objective must be and is represented as a force applicator.

These representations are posited to apply to general design domains. In this chapter they will be applied to the specific design domain of architectural space planning, and with slight modification of nomenclature to general floor planning problems. A prototype computer application that provides a user interface for these concepts is described in the next section. Although the behavior of these elements in action can be imagined, it is only through experiencing them in a real-time setting that the benefits of the proposed approach emerge.

Figures 4-1 and 4-2 give some idea of where the concepts described in this section will take us, and show the interesting relationship between the space planning problem as seen through the eyes of the designer and the same problem as seen through the 'eyes' of the dynamical system. Figure 4-1 shows the interior walls of each space in a sample result of a simple space plan, while figure 4-2 shows the same problem at the same solution state with masses representing space centers and walls, and lines between these masses representing a design objective modeled as a force applicator. This section defines elements in one system, the dynamical system

in figure 4-2, in such a way that they support the creation of elements in another system, the mental and graphical design system in figure 4-1.

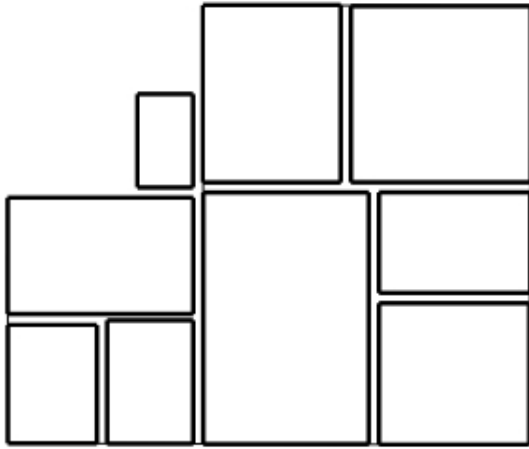


Fig. 4-1. Designer's problem

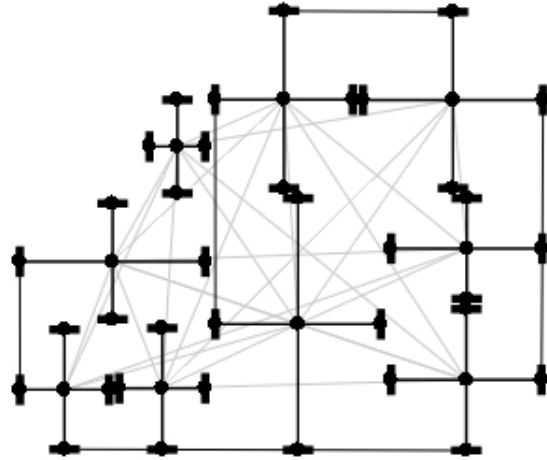


Fig. 4-2. Dynamics problem

4.1. Design elements

Physically based space planning design elements are defined to represent the spaces and walls of a building in such a way that they can be used in a physically based simulation. A configuration of design elements defines the state of a building design at a specific instant in time, while design objectives change the design state over time.

At a minimal level the only design element required in space planning is a *Space*. The representation of a space requires a *shape* to define its boundaries, and a mass for the entire space as well as for each of its individual edges. *Nodes* represent masses, on which design objective forces will be applied.

For example, a space can be represented as a polygonal boundary with a point

mass at its center. The boundary can be represented as an ordered series of masses located on edges that maintain their orientation, and alternatively can be represented as masses located on each boundary vertex. With this representation, forces can be applied to the masses, inducing them to change position.

4.1.1. Nodes

A node is a point in space on which a force can be applied. The data structure representing a node contains values for mass, position, and velocity, as well as a force accumulator and other geometric information that may be required for each node type. Each unique node type has its own graphic representation. Nodes are typically connected to other nodes by springs. The type of the node determines how its movement, and the movement of the node to which it is connected, is constrained.

The basic data structures used within nodes are a point

```
class Point
    x, y, z: Real,
```

which defines a position in a three-dimensional coordinate system, and a vector

```
class Vector
    x, y, z: Real,
```

which defines a magnitude and a direction in the same coordinate system.

4.1.1.1. Point node

A point node is the simplest node type. The data structure for a point node stores the following typical node information

```
class Point Node
    mass      : Real
    position  : Point
    velocity  : Vector
    force     : Vector,
```

and is displayed as a dot. A force applied to a Point Node is not constrained in any way. Figure 4-3 shows a point node with position $\bar{x}(t)$ at time t , with a constant force \mathbf{f} applied to it, and no initial velocity. A point node at position $\bar{x}(t)$ is accelerated in the direction of \mathbf{f} so that at time $t + \Delta t$ it is at position $\bar{x}(t + \Delta t)$.

Point nodes are typically used to define the centers of spaces.

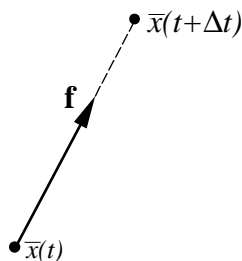


Fig. 4-3. Point node

4.1.1.2. Line node

A line node defines an infinite line passing through a point. Any force applied to a line node is constrained to act perpendicular to the line it defines, thus preserving its orientation. The data structure for a line node, which contains the typical node information as well as unit direction and unit normal vectors, is

```
class Line Node
  mass      : Real
  position  : Point
  velocity  : Vector
  force     : Vector
  direction : Vector
  normal    : Vector.
```

A position and a direction are all that are needed to define a line; the additional normal vector is stored so as not to repeat its calculation. A line node is displayed

as a dot with a short bar going through it parallel to the direction vector. Figure 4-4 shows a line node with unit direction \mathbf{d} , unit normal \mathbf{n} , and position $\bar{x}(t)$ at time t , with a constant force \mathbf{f} applied to it, and no initial velocity. The line is constrained to move along \mathbf{n} by applying

$$\mathbf{f}' = (\mathbf{f} \cdot \mathbf{n})\mathbf{n},$$

the component of the force in the direction normal to the line. In the preceding equation the \cdot operator is the dot product of two vectors. Thus, a line node at position $\bar{x}(t)$ is accelerated in the direction of \mathbf{n} so that at time $t + \Delta t$ it is at $\bar{x}(t + \Delta t)$.

Line nodes are typically used to define the polygonal edges of space boundaries.

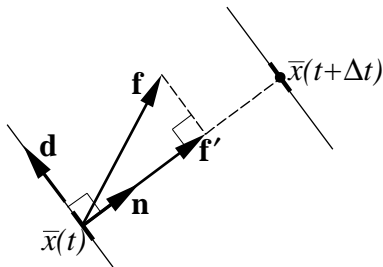


Fig. 4-4. Line node

4.1.1.3. Plane node

A plane node defines an infinite plane passing through a point. Any force applied to a plane node is constrained to act perpendicular to the plane it defines, thus preserving its orientation. The data structure for a plane node, which contains the typical node information as well as a unit normal vector, is

```

class Plane Node
  mass      : Real
  position  : Point
  velocity  : Vector
  force     : Vector
  normal    : Vector.

```

A plane node could be displayed as a dot with a square drawn around the dot, oriented to the plane. Figure 4-5 shows a plane node with unit normal \mathbf{n} , and position $\bar{x}(t)$ at time t , with a constant force \mathbf{f} applied to it, and no initial velocity. The plane is constrained to move along \mathbf{n} by applying

$$\mathbf{f}' = (\mathbf{f} \cdot \mathbf{n})\mathbf{n}.$$

the component of the force in the direction normal to the plane. Thus, a plane node at position $\bar{x}(t)$ is accelerated in the direction of \mathbf{n} so that at time $t + \Delta t$ it is at $\bar{x}(t + \Delta t)$.

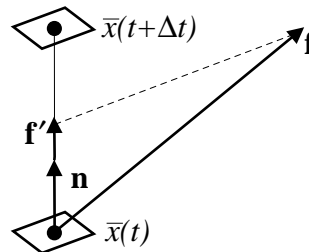


Fig. 4-5. Plane node

It would seem natural to use a plane node in three-dimensions to represent a wall or one face of a space volume. However, due to the nature of the architectural space planning problem, extending space planning to the third dimension is not that straightforward. This is discussed in section 9.2.1.2. A building in three dimensions

contains one or more plans that by themselves remain in two dimensions, and there is little to be gained by using space volumes and plane nodes.

Despite the seeming lack of usefulness of a plane node in space planning, this element could be useful in other three dimensional design domains, such as building massing design or manufactured artifact design.

4.1.2. Polygons

Now that the basic node elements are defined, they can be used to define and build up more complex elements.

A *polygon* is a closed plane figure with straight boundary lines, and is defined here as an ordered list of a pair of line nodes and polarities, where each line node is connected to a common center node with a line perpendicular to the direction vector of the line node. No two edges of a valid polygon can cross each other. Examples are shown in figure 4-6.

Figure 4-6a shows an arbitrary n -sided polygon, with center node c , edge line nodes e , and vertices \bar{v} . Each vertex \bar{v}_i can be found as the intersection of edges e_i and $e_{i\oplus 1}$, where the operator \oplus is addition modulo n . This representation of a polygon can also be found in Kalay (1989, p. 36). The *polarity* of an edge defines the direction that the next edge will turn. The convention used here is -1 to the right, and 1 to the left. For example, the polarity list of the polygon in figure 4-6a is (1, 1, -1, 1, 1, 1). Given this representation, the data structure for a polygon is

```
class Polygon
    edges      : Line Node list
    polarities : Integer list [-1, 1].
```

The position of an edge along its direction vector technically does not matter; all positions along the direction vector yield the same line, and thus the same polygon.

Therefore, an infinite combination of line nodes can yield the same polygon. Even when given the additional constraint that each line node must be connected to a common center node with a perpendicular line, there are still an infinite combination of line nodes that define a single polygon, which can be shown by comparing figure 4-6a with figure 4-6b.

The polarity list insures that the topology of the polygon does not change. Using only a line node list, the two polygons in figures 4-6b and c produce the same polygon. The addition of a polarity list differentiates them into two different polygons.

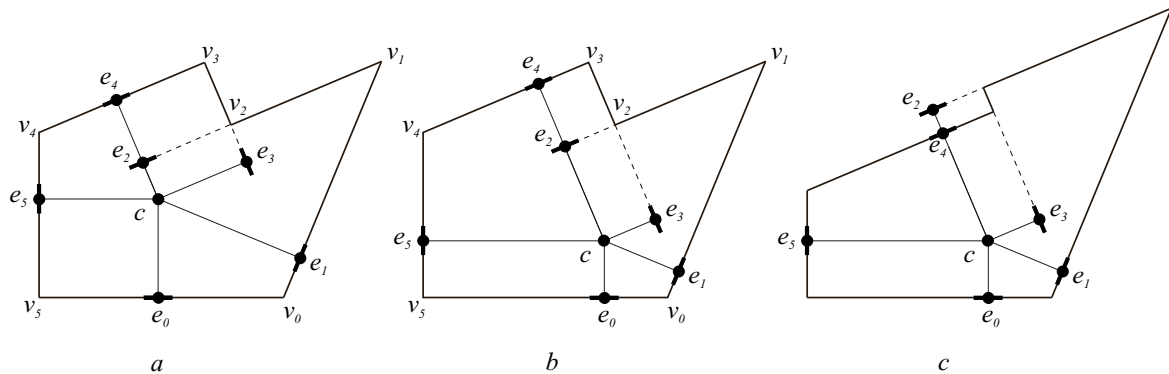


Fig. 4-6. An arbitrary polygonal shape represented with edge line nodes

This may seem like an overly complicated way to represent a polygon, and that a simple list of vertices would suffice. There are three reasons to choose this representation. 1) It allows the orientation of each edge to be easily maintained; 2) it allows a connection to be made between each edge node and a center node, which will be used later as the center of a space; and 3) it allows the topology of the polygon to be fixed; that is, it prevents the edges from ‘flipping.’

Easy maintenance of edge orientation is needed if a polygon is used to define

the boundary of a space. It allows us to apply forces to individual segments in such a way that the edge moves but does not rotate. A typical task in space planning is to move a wall, not to move the end points of a wall. If a shape were represented with point nodes at its vertices, in order to maintain edge orientation any force applied to one vertex would have to be separated into components that are applied to its surrounding vertices. For non-orthogonal shapes, computing the necessary components could get unnecessarily complicated. Figure 4-7a shows a rectangle represented with vertex point nodes. A force applied to one of its vertices as shown results in the non-orthogonal polygon shown in figure 4-7b. Figure 4-7c shows a rectangle represented with edge line nodes. Forces \mathbf{f}_1 and \mathbf{f}_2 applied to its edges yield resultant forces \mathbf{f}'_1 and \mathbf{f}'_2 , which act normal to each edge, resulting in the maintained rectangular shape shown in figure 4-7d.

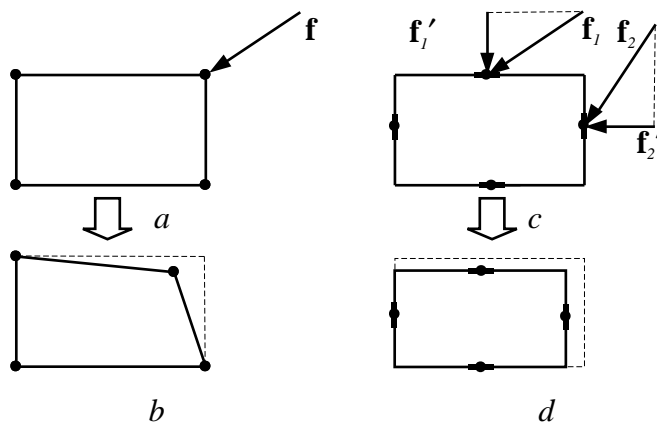


Fig. 4-7. A rectangle represented with vertex point nodes and edge line nodes

Vertex polygonal representations are potentially useful in modeling soft or amorphous shapes, or a number of soft or amorphous edges on an otherwise rigid shape.

A space plan consisting of totally fluid shapes would probably result in plans looking like cell membranes or other organic forms (Thompson, 1992; Prusinkiewicz and Lindenmayer, 1990), such as that shown in figure 4-8. A single space containing some soft edges and some rigid edges is potentially useful in allowing such a space to wrap around another rigid shape, as shown in figure 4-9, where part of the almost rectangular shape wraps around the circular shape.

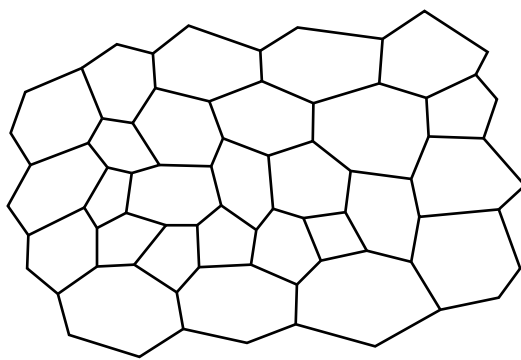


Fig. 4-8. A space plan using vertex polygonal representations [From Prusinkiewicz and Lindenmayer (1990, p. 154)]

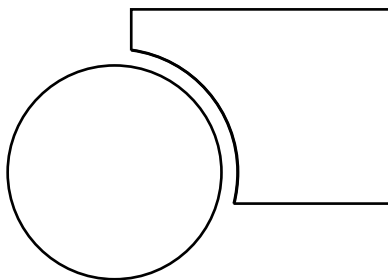


Fig. 4-9. A space with soft edges wrapping around a space with rigid edges

4.1.3. Spaces

A space defines any arbitrary polygonal area or the volume of a polygonal extrusion. The data structure representing a space, which contains a common center node to define its location and a polygon to define its boundary, is

```
class Space
  center : Point Node
  shape  : Polygon.
```

The shape value is optional, to enable the definition of any generalized space. For example, a space used to represent the outside would not need a defined position or shape, and could be used when a building space needs to relate to the outside.

A hierarchy of spaces of arbitrary depth can be constructed by allowing any space to contain any number of child spaces, each of which can contain their own set of child spaces, as shown in figure 4-10a. A similar structure is described in Flemming and Chien (1995). A parent space and its child spaces define a self-contained physical system, and the relationship between the parent and its children is defined by the parent boundary. A set of spaces that all have the same parent space will be referred to as a *sibling* set. If a parent boundary exists, as in figure 4-10a, the system of child spaces needs to be contained within that boundary. If a parent boundary does not exist, as in figure 4-10b, the union of the child space boundaries will define the parental boundary.

The location of the center node relative to the edges is not important to the polygonal representation, as described in section 4.1.2, but is important in determining space adjacencies. For simple shapes the geometric center or center of mass is fine, but for more complicated shapes a more appropriate center may need to be defined by the designer. For example, a U-shaped space might have its center of mass located outside the boundary, but the designer might prefer that the center

node be located inside the boundary.

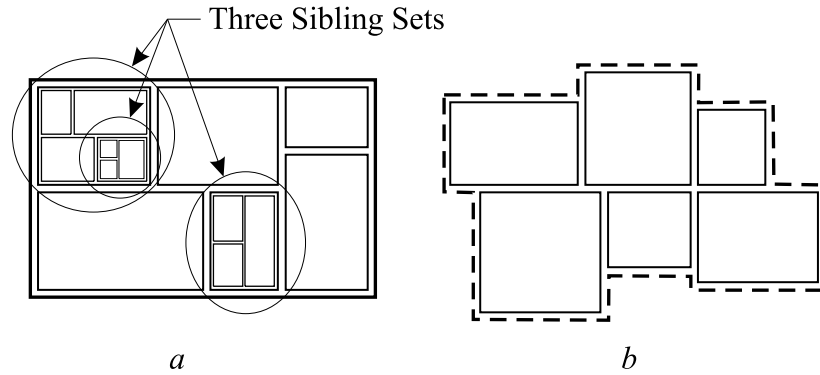


Fig. 4-10. Parent space with and without a defined boundary

4.1.3.1. Local coordinate system

The polygonal shape for a space is defined and maintained in the local coordinate system of the space. Each space has its own local coordinate system with its center node always located at the origin. The center node contains the space's position in world coordinates, while each polygonal shape contains wall node positions in the space's local coordinates. The vertices used to draw the shape are computed and temporarily stored in world coordinates. Forces applied to a Space Node have no direct effect on anything stored in the Space's local coordinate system, while forces applied to a Wall Node result in changes to the Space's polygonal shape. Figure 4-11 describes what happens in a sample problem when a force is applied to a Wall Node.

An obvious question to ask is if moving a wall ends up affecting the position of the space, then why not just move the space? The answer is that moving a space would be the same as moving all walls, with the result that the space's shape can

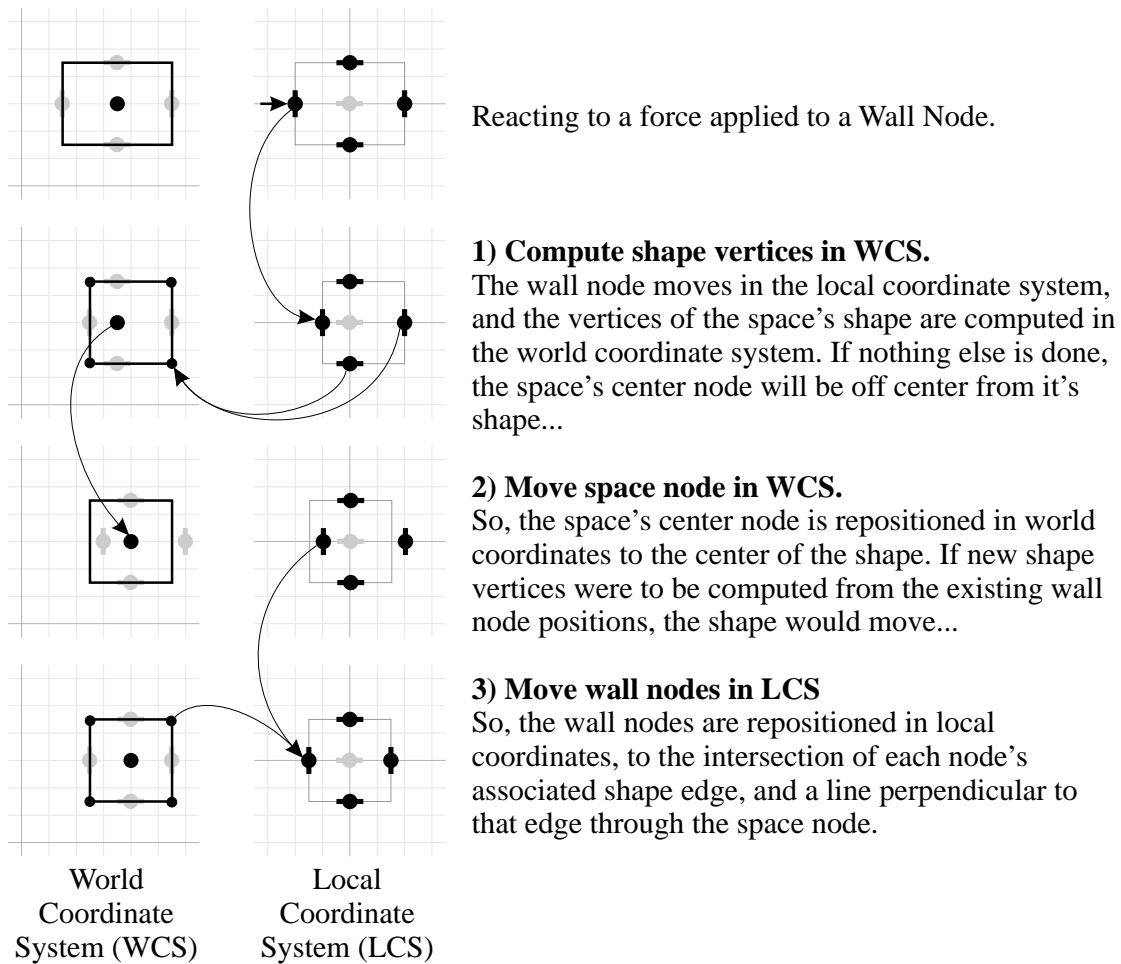


Fig. 4-11. Use of a space local coordinate system

never *change*. By only moving a single wall, different forces may be moving other walls, thereby changing the shape of the space. The position of the space itself is affected by the aggregate of *all* wall forces.

4.2. Design objectives

While design *elements* define the state of a design at a specific instant in time, design *objectives* define the intentions that the designer may have toward changing

that state. The primary requirement of a design objective in a physically based design approach is to apply forces to the spaces and walls represented in the data structures. These applied forces should act to change the location of the spaces and walls, thereby changing the design over time. Additionally, they need to be defined in terms that are familiar to designers, and they need to cause design elements to behave as expected.

4.2.1. Properties of design objectives

Each type of design objective has a number of properties that define how such objectives act. Each design objective defines the type and number of nodes on which it acts (represented in formulas with their position \bar{p}), the desired configuration of those node positions as a scalar d or vector \mathbf{d} , a method of computing the configuration error e or \mathbf{e} , and a level of importance I . These values vary for each instance of a design objective. A strength constant k_{type} is defined for all instances of a type of design objective. The configuration error, level of importance, and strength constant are multiplied to compute the configuration force \mathbf{F}_{type} that will be applied to the individual nodes to move them from an undesired configuration to a desired configuration. These properties are summarized in table 4-1.

4.2.1.1. Nodes

Each design objective specifies one or more *nodes* on which it will act. The node types and the manner in which they are specified are determined by the type of design objective. A design objective that can be applied to either wall nodes or space nodes specifies one or more nodes. An objective that can be applied only to wall nodes specifies one or more line nodes. An objective that can be applied to space nodes specifies a space, and the objective itself accesses the space's center node. An

Table 4-1. Properties of a design objective

Property	Specification	Term
Node(s)	Node (if type does not matter) Line Node or Point Node (if type matters) Space Node Space Polygon	\bar{p} , or \bar{p}_i for i^{th} node
Desired Configuration	Scalar Vector	d , for example \mathbf{d} , for example
Configuration Error	Vector	\mathbf{e} , \mathbf{e}_i
Importance	Real [0.0 - 1.0]	l
Strength Constant	Real	k_{type}
Configuration Force	Vector	\mathbf{F}_{type}

objective that can be applied to a space polygon specifies a space, and the objective itself would access the space's polygon's nodes.

The position term of a node is \bar{p} , or \bar{p}_i for the i^{th} node in a list of nodes.

4.2.1.2. *Desired configuration*

Each design objective specifies a numeric value that indicates the *desired configuration* of the nodes. Each type of design objective specifies the meaning of that numeric value, such as a scalar distance d or area a , or a vector direction \mathbf{d} .

4.2.1.3. *Configuration error*

Each type of design objective specifies a way of computing the *configuration error* of each of the nodes. Configuration error is defined to be a measure of the difference between a desired configuration and an undesired configuration. If the current configuration *is* the desired configuration, then there is no error. If the current configuration *is not* the desired configuration, the magnitude of the configuration error is proportional to the degree with which the current configuration is not the desired configuration.

The term for the configuration error is \mathbf{e} , or \mathbf{e}_i for the configuration error computed for the i^{th} node.

4.2.1.4. *Level of importance*

Each design objective specifies the *level of importance* of one instance of a design objective relative to another instance, possibly of a different type. This is a numerical value that scales the magnitude of the configuration error. For example, if an exterior objective applied to one space is more important than an exterior objective applied to another, their relative level of importance can be set accordingly.

Level of importance is set within the range (0.0 - 1.0]. Sanoff describes a similar method of using a numerical value to define a relationship:

This discussion is based on binary decisions between two activities; either there is a relationship or there is not. It is possible to assume that there is some connection between all the activities in the matrix so that the important distinction is the magnitude of the dependency between each pair. In this case it would be appropriate to substitute a numerical scale value from one to five to indicate potency, rather than a notation to indicate dependency. (Sanoff, 1977, p. 109)

A set of descriptive terms can be used to help define numeric values for levels of importance, as in the sample set shown in table 4-2.

The term for the level of importance is I_{type} .

4.2.1.5. *Strength constant*

Each type of design objective specifies a *strength constant* applied to all instances of one type of design objective. This is a numerical value that scales the magnitude of

Table 4-2. Sample levels of importance

Descriptive requirement	Numerical value
Mandatory	1.0
Significant	0.6
Desirable	0.3
If Possible	0.1

the configuration error, similar to level of importance.

The purpose of the strength constant is to scale the effect of the design objectives so that objectives with the same level of importance maintain the same relative strength to each other. Each type of design objective is implemented in a slightly different way, and this differing implementation sometimes results in one type being apparently stronger or weaker than another. An appropriate set of design objective strength constants should result in two objectives of differing types but with the same level of importance having the same apparent strength.

The term for the strength constant for a type of design objective is k_{type} , such as k_{ad} for an adjacency objective.

4.2.1.6. Configuration force

The configuration force is the force actually applied to a specific node in a physically based space planning system. It is simply the configuration error scaled by both the level of importance and the strength constant.

The term for the configuration force is \mathbf{F} , or \mathbf{F}_i for the force applied to the i^{th} node, and is

$$\mathbf{F} = k_{type} I_{type} \mathbf{e}.$$

Table 4-3. Classes of objectives and their properties

	Topological	Geometric
Unary	Move one space	Move all walls of one space, without necessarily moving the space itself
Binary	Move two spaces	Move two walls from different spaces

4.2.1.7. *Description*

In the descriptions for the types of design objectives that follow, only the properties for nodes, desired configuration, and configuration error will be described. These properties vary from objective to objective and define how one is applied differently from others. The importance and strength constant properties are defined in the same way for all types, and given all of these properties the configuration force is computed in the same way for all types.

4.2.2. *Classes of design objectives*

Design objectives are grouped here according to their similarity to each other. They can be designed to change the relationship between design elements or to change the geometry of design elements. They can act on a single element or on multiple elements. The possible combinations and a description of how they affect design elements in a space plan are shown in table 4-3.

4.2.2.1. *Topological vs. geometric*

The space planning problem is often separated into two sub-problems (Jo and Gero, 1998; Flemming, 1989). The first problem is satisfying topological properties such as the location of individual spaces relative to each other. Some topological objectives described here include adjacency, separation, orientation, and interior-exterior objectives. The second problem is satisfying geometric properties such as the size and

Table 4-4. Physical analogues of design objectives

Unary Topological Objectives	
Interior	Spring
Exterior	Spring
Orientation	Spring
Binary Topological Objectives	
Adjacency	Spring
Separation	Repulsion Field
Unary Geometric Objectives	
Area	Balloon
Proportion	Clay
Binary Geometric Objectives	
Alignment	Screw clamp

shape of space boundaries. Some geometric objectives described here include area, alignment, and proportion objectives.

All topological objectives minimally specify a space whose location the objective is trying to influence, and a vector, which specifies the direction and magnitude of the force being applied. The differences among the various topological objectives lie in the manner in which the direction vector is specified.

Geometric design objectives influence the dimensions of space boundary edges. Any objective that results in a force being applied to an edge is a geometric objective. All geometric objectives minimally specify a node or set of nodes whose location the objective is trying to influence.

4.2.2.2. *Unary vs. binary*

Unary objectives apply a single force to a single element. Binary objectives apply a pair of forces, equal in magnitude and opposite in direction, to two different elements.

4.2.3. Physical analogues

Since spaces and walls are represented with nodes that have a mass, they are analogous to physical objects in the real world. Design objectives, which apply forces to nodes, also have physical analogues. Table 4-4 lists the design objectives that will be described in the rest of this section, along with their physical analogue.

4.2.4. Unary topological objectives

Unary topological objectives apply a single force to a single space node. The intent of a unary topological objective is to move a single space in the direction specified by the force vector, so these objectives will also be called direction objectives. The differences among the various direction objectives are in the manner in which the direction vector is specified and how it may change. Unary topological objectives tend to be mutually exclusive, so a single space should in general have only one specified at any one time.

There are two additional considerations unique to direction objectives that affect their design. All of the design objectives that will be described later are balanced, in that they define equal and opposite forces. They are also easily bounded, in that the minimum and maximum force magnitudes are fairly obvious. Because direction objectives apply a single force to a single space, they are unbalanced. In the absence of other counteracting forces, the force applied by a direction objective to a plan will cause it to continually move.

As stated in section 4.2.1, a good design objective needs to apply a force that is proportional to the degree to which it is not met. Defining a lower bound of this force, the point at which the objective has been met and the magnitude of the force is zero, is critical. In order to find the bounds of a direction objective force, two

conditions must be defined: what is the ‘interior’ and what is the ‘exterior’ of a set of spaces. A variety of methods can be used to define these conditions.

4.2.4.1. Interior objective

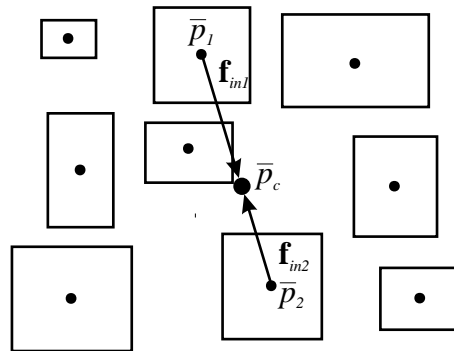


Fig. 4-12. Interior objectives

An interior objective is a topological objective that attempts to locate a space in the center of its set of sibling spaces. For example, an interior objective may be used for spaces that have privacy or security requirements that may be better achieved by being located away from exterior walls.

The data structure for an interior objective contains only a space

```
class Interior Objective
    space      : Space      (space center node)
    (direction is computed) (desired configuration).
```

The space’s center node is used for the nodes property. A desired configuration need not be defined because the direction and magnitude of the configuration error can be computed from the position of the space within it’s set of siblings.

The *parental center* \bar{p}_c of a space is defined here as the average position of it

and each of its sibling's positions. For n spaces with space node position \bar{p}_i

$$\bar{p}_c = \left(\sum_{i=1}^n \bar{p}_i \right) / n.$$

This is just one of a variety of methods that can be used to define the parental center.

The configuration error is the vector from the space center \bar{p} to the parental center

$$\mathbf{e} = \bar{p}_c - \bar{p}.$$

The interior objective configuration force \mathbf{F} is

$$\mathbf{F}_{in} = k_{in} I_{in} \mathbf{e}.$$

Its magnitude is proportional to the distance between the space center node and the parental center, and is zero when the space node is at the parental center. This force is applied in the direction of the direction vector, and tends to move the space toward the parental center and away from the parental boundary.

Figure 4-12 shows a set of sibling spaces with parental center \bar{p}_c . Two of the spaces have interior objectives specified, shown with configuration forces \mathbf{F}_{in1} and \mathbf{F}_{in2} pointing toward the parental center \bar{p}_c .

4.2.4.2. Exterior objective

An exterior objective, the opposite of an interior objective, is a topological objective that attempts to locate a space toward the boundary edges of its parental shape. For example, an exterior objective may be used to specify that a space has daylighting requirements.

The data structure for an exterior objective is the same as that for an interior objective and contains only a space

```
class Exterior Objective
```

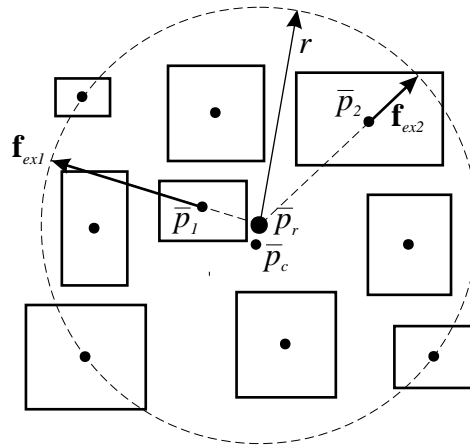



Fig. 4-13. Exterior objectives

space : Space (space center node)
 (direction is computed) (desired configuration).

The space's center node is used for the nodes property. A desired configuration need not be defined because the direction and magnitude of the configuration error can be computed from the position of the space within it's set of siblings.

As described previously, a good design objective needs to apply a zero magnitude force once the objective has been met. An exterior objective has been met when the space is on the 'exterior' of its set of sibling spaces. The 'exterior' is a fairly loose term that can be hard to define in some contexts. Given a floor plan, a room is on the 'exterior' if one of its walls is an exterior wall. As will be described later, during topological resolution spaces are treated as circles. What is the 'exterior' of a set of circles? Is it enough to find the union of all of the circles, and if one of the exterior arcs on that union is contributed by the subject space is it considered to be on the exterior? But what if, when geometric resolution is solved, the space is no longer on the exterior?

As with defining the parental center \bar{p}_c of a set of spaces, described in the

previous section, there are a variety of ways to define the ‘exterior’ of a set of spaces, both in a circular and polygonal representation of space boundaries. The method described here is not ideal, but is used to demonstrate a relatively simple approach.

To find the exterior of a set of spaces, and thus an exterior point with which to measure an exterior objective, find the smallest enclosing circle that contains all sibling space centers, as shown in the circle with center \bar{p}_e (hereafter called the enclosing center) and radius r in figure 4-13. Note that the enclosing center is not equal to the parental center.

Let \mathbf{d} be a vector from the enclosing center \bar{p}_e to the space center \bar{p}

$$\mathbf{d} = \bar{p} - \bar{p}_e.$$

The direction vector

$$\mathbf{d}_{ex} = \frac{\mathbf{d}}{|\mathbf{d}|}$$

is a unit vector in the direction of \mathbf{d} .

The strength s needs to be proportional to the distance between the space center and the ‘exterior’ of its set of sibling spaces, so it is the radius of this circle r minus the distance from the enclosing center to the parental center

$$s = r - |\mathbf{d}|.$$

The configuration error is then the strength times the unit direction

$$\mathbf{e} = s\mathbf{d}_{ex}.$$

The exterior objective configuration force \mathbf{F}_{ex} is

$$\mathbf{F}_{ex} = k_{ex}I_{ex}\mathbf{e}.$$

If the space center coincides with the parental center the direction vector is undefined as described above, so it should point toward the nearest edge of the bounding box surrounding the sibling centers. If the direction vector is still undefined due to equal distances to each bounding box edge, a random direction should be chosen.

The force \mathbf{F}_{ex} is applied in the direction of the direction vector, and tends to move the space away from the enclosing center and toward the parental boundary. Two of the spaces in figure 4-13 have exterior objectives specified, shown with configuration forces \mathbf{F}_{ex1} and \mathbf{F}_{ex2} pointing away from \bar{p}_c .

4.2.4.3. Orientation objective

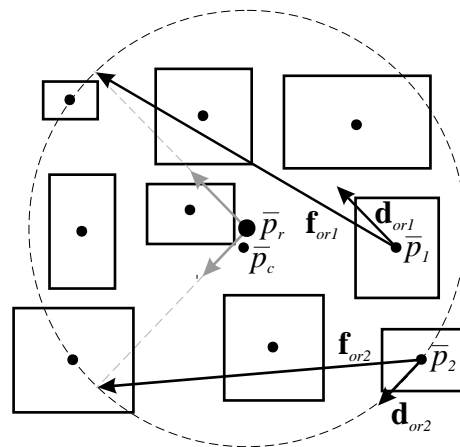


Fig. 4-14. Orientation objectives

An orientation objective influences where a space is located on the building perimeter. See Akin et al. (1988, p. 417) for a similar use of orientation. For example, an orientation objective can be used to specify a particular view for an office, or that

a public parking area needs to be near the site entrance.

The data structure for an orientation objective contains a space, and a direction vector

```
class Orientation Objective
    space      : Space      (space center node)
    direction  : Vector     (desired configuration).
```

The space's center node is used for the nodes property. The desired configuration is the direction from the center of the set of sibling spaces to the point on the outside of this set where the subject space is desired to be.

The direction vector for an orientation objective points to the side of the set of sibling spaces where the space needs to be located. It can be specified with an angle, a vector, or with descriptive terms such as Northeast or Southwest (see appendix B). Unlike interior and exterior objectives, whose direction vectors can change continuously throughout a dynamics simulation, the direction vector for an orientation objective is specified by the user when the objective is defined, and remains constant throughout the simulation. As with the other direction objectives, there are a variety of methods that can be used to define the configuration force; the method described here is just one.

Given a specified unit direction vector \mathbf{d}_{or} , and an enclosing circle with enclosing center \bar{p}_e and radius r (as described in the previous section), the exterior point \bar{p}_x on the enclosing circle in the direction of the direction vector is

$$\bar{p}_x = \bar{p}_e + r\mathbf{d}_{or}.$$

The configuration error is the vector from the space center \bar{p} to the exterior point \bar{p}_x

$$\mathbf{e} = \bar{p}_x - \bar{p}.$$

The orientation objective configuration force \mathbf{F}_{or} is

$$\mathbf{F}_{or} = k_{or} I_{or} \mathbf{e}.$$

Two of the spaces in figure 4-14 have orientation objectives specified; one shown with Northwest direction vector \mathbf{d}_{or1} and its resulting force \mathbf{F}_{or1} , and another shown with Southwest direction vector \mathbf{d}_{or2} and its resulting force \mathbf{F}_{or1} . These forces tend to move the spaces to the side of the building specified by the direction vector.

4.2.4.4. *Applications and relationships*

To summarize the relationships among these objectives, an interior objective points toward the center of a group of spaces, an exterior objective points away from the center of a group of spaces, and an orientation objective always points in a specific direction. The direction of interior and exterior objectives can change with time, depending on the location of the associated space relative to its siblings.

Architects might use these objectives to meet a variety of design objectives. For example, if an architect wants a particular space to have daylight, she might use an exterior objective to position the space on the exterior of the building, or if she is concerned about sunlight or the quality of light in the space she might use an orientation objective to locate the space on a specific side of the building. For another example, if a building site has a particularly good view in one direction the architect might want the windows in a space oriented toward that view, so he might use an orientation objective pointing in the direction of the view.

There is not a one-to-one relationship between the designer's objectives and the technical implementation of those objectives. An orientation objective cannot simply be renamed to a daylight objective because an exterior objective can be used to achieve the same goal. Also, that same renaming hides the fact that an orientation

objective can be used to accomplish a view design objective as well as daylight. The design of the user interface for a practical application, if following the goal-directed design process specified in Cooper (1999), should follow the needs of the designer instead of the needs of the implementation. It might be acceptable to have interior, exterior, and orientation objectives, as implemented, but users would probably want an explicitly named daylight or view objective. The view objective would simply be an orientation objective with another name, while the daylight objective would need to be implemented as either an exterior or an orientation objective, depending on the specific needs of the designer for a specific space. The need is to capture and label the designer's intent as well as to implement that intent. An exterior objective says nothing about *why* the space needs to be on the exterior, while labeling the objective daylight or view does. This enables the designer to later make appropriate design modifications. For example, if it is later determined that it is not necessary for a space to have a view, the objective can be removed. If the objective was called exterior, however, there might be other non-view related reasons for having it, and the designer might be hesitant in removing it.

4.2.5. *Binary topological objectives*

Binary topological objectives apply a force to two space nodes. The intent of a binary topological objective is to change the location of two spaces relative to each other. The data structure for a binary topological objective minimally contains two spaces

```
class Binary Topological Objective
    space1 : Space
    space2 : Space.
```

4.2.5.1. Adjacency objective

An adjacency objective is a topological objective that attempts to locate two spaces next to each other. For example, two spaces that have a large amount of traffic between them may need to be located very close together. Figure 4-15 shows two spaces connected with an adjacency objective, and how a system of spaces connected with a number of adjacency objectives might be resolved.

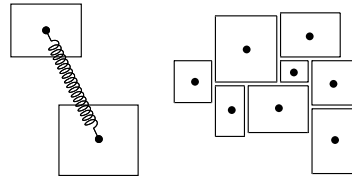


Fig. 4-15. Adjacency objective

The data structure for an adjacency objective contains two spaces

```
class Adjacency Objective
    space1      : Space      (space center node)
    space2      : Space.
```

Each space's center node will be used for the nodes property. The desired configuration is that the distance between the spaces be zero.

The center nodes of each space are connected with a spring (see section 3.3.3.2), which applies forces to the nodes depending on the distance between them. If the spaces are too far apart, the spring will produce forces on each space node that attempt to move them together, and vice versa.

The vector from *space1*'s center \bar{p}_1 to *space2*'s center \bar{p}_2 is

$$\mathbf{d}_{12} = \bar{p}_2 - \bar{p}_1.$$

When using the circular representation of spaces, the desired configuration, or rest length, of the spring is the sum of the radii of each space

$$r = r_1 + r_2.$$

The configuration error on *space1* is

$$\mathbf{e}_1 = -(|\mathbf{d}_{12}| - r) \frac{\mathbf{d}_{12}}{|\mathbf{d}_{12}|}.$$

While the configuration error on *space2* is the opposite of that of *space1*

$$\mathbf{e}_2 = -\mathbf{e}_1$$

The adjacency objective configuration force \mathbf{F}_{ad} for each space is

$$\mathbf{F}_{ad1} = k_{ad} I_{ad} \mathbf{e}_1$$

$$\mathbf{F}_{ad2} = k_{ad} I_{ad} \mathbf{e}_2.$$

4.2.5.2. Separation objective

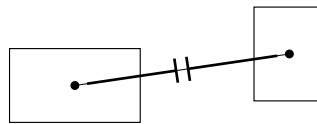


Fig. 4-16. Separation objective

A separation objective is the opposite of an adjacency objective, and is a topological objective that attempts to locate one space away from another. Figure 4-16 shows two spaces connected with a separation objective. For example, a separation objective may be used to keep a private space and a public space on opposite sides of

a building. Separation objectives can be problematic, in that two spaces may need to be separated from each other, but not so much that the system they belong to separates into two different sets.

A separation objective is designed as a repulsion field that repels one space away from another when they are located too close together. The data structure for a separation objective contains two spaces, and the minimum distance required between the edges of each space

```
class Separation Objective
    space1      : Space
    space2      : Space
    distance    : Real.
```

A repulsion field is in a sense a spring with a rest length equal to the minimum separation distance, but that only expands and never contracts, because a repulsion field ‘spring’ applies no force when the distance between the spaces is greater than the minimum distance.

The unit vector from *space1* to *space2* \mathbf{d}_{12} is

$$\mathbf{d}_{12} = \frac{\bar{p}_2 - \bar{p}_1}{|\bar{p}_2 - \bar{p}_1|}.$$

Using a circular representation for the area of each space, the separation distance d between the two spaces is the distance between each space’s center node minus the sum of their radii

$$d = |\bar{p}_2 - \bar{p}_1| - (r_1 + r_2).$$

If $d > d_{se}$, where d_{se} is the minimum separation distance defined for the separation objective, then the configuration errors for each space, \mathbf{e}_{se1} and \mathbf{e}_{se2} , are zero.

Otherwise, they are

$$\mathbf{e}_1 = -(d_{se} - d)\mathbf{d}_{12}$$

$$\mathbf{e}_2 = -\mathbf{e}_1.$$

The direction of the error for *space1* is away from *space2*, and vice versa.

The separation objective configuration forces for each space are

$$\mathbf{F}_{se1} = k_{se}I_{se}\mathbf{e}_1$$

$$\mathbf{F}_{se2} = k_{se}I_{se}\mathbf{e}_2.$$

4.2.6. Unary geometric objectives

Unary Geometric Objectives apply forces to all of the edge nodes of a single space. The intent of a unary geometric objective is to change the dimensions of a single space by changing the dimensions of each of its edges. The data structure for unary geometric objectives minimally contains a space, a target value, and a range

```
class Unary Geometric Objective
  space      : Space
  target     : Real
  range
    percent  : Real
    or
    minimum  : Real
    maximum  : Real.
```

The meaning of the target and range values are set by each individual objective.

4.2.6.1. Area objective

An area objective is a geometric objective that attempts to maintain a specified area for a polygonal shape. Figure 4-17 shows a rectangular polygon, with the desired area shown as a dashed rectangle, the area range shown as dotted rectangles, and

area objective forces applied to the edge line nodes. The forces applied by an area objective are analogous to a balloon in that they are applied to the insides or outsides of all edges of the polygonal shape.

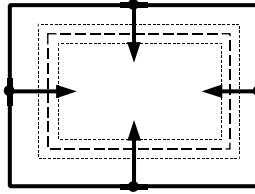


Fig. 4-17. Area objective

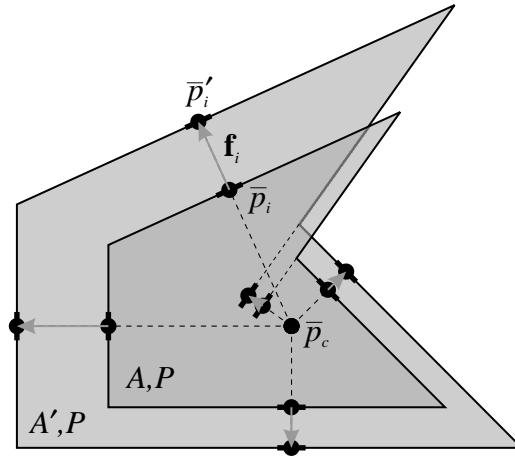


Fig. 4-18. Area objective forces

Area objective forces need to be defined in such a way that they change the area but not the proportion of the shape they are applied to. Figure 4-18 shows a space's polygonal shape with current area A and proportion P and its intended

shape with target area A' and the same proportion. The point \bar{p}_c is the position of the space's center node, \bar{p}_i is the current position of one of the edge nodes of the polygon defining the space, \bar{p}'_i is the target position of the same node, and \mathbf{f}_i is the force vector needed to move \bar{p}_i toward \bar{p}'_i . A polygon's proportion does not change after it has been scaled. The scale value s necessary to scale a polygon with area A to area A' is

$$s = \sqrt{\frac{A'}{A}}.$$

Each point \bar{p}'_i is found by scaling \bar{p}_i from \bar{p}_c by s

$$\bar{p}'_i = \bar{p}_c + s\bar{p}_i.$$

Since the position of a space's wall nodes are stored in the space's local coordinate system with its center node at the origin, as described in section 4.1.3.1, \bar{p}_c is the origin, and thus \bar{p}'_i becomes

$$\bar{p}'_i = s\bar{p}_i.$$

The configuration error between \bar{p}_i and \bar{p}'_i for each edge node is then

$$\begin{aligned} \mathbf{e}_i &= s(\bar{p}_i) - \bar{p}_i \\ &= (s - 1)\bar{p}_i. \end{aligned}$$

As area A approaches A' , s approaches 1, and each of the errors \mathbf{e}_i approach zero.

The area objective configuration force for each line node i is

$$\mathbf{F}_{ar\ i} = k_{ar}I_{ar}\mathbf{e}_i.$$

If an area range is defined for the area objective as described above, then s also depends on the target area range. If the current area is too small ($A < A_{min}$), then $A' = A_{min}$, while if it is too large ($A > A_{max}$), then $A' = A_{max}$. If the current area

is within the desired range ($A_{min} < A < A_{max}$), then $A' = A$ and $s = 1$, resulting in no needed change in area and thus no forces applied to the nodes.

4.2.6.2. Proportion objective

A proportion objective is a type of geometric objective that attempts to maintain the specified proportions of a polygonal shape. It is similar in concept and application to the area objective. If the proportions of a polygon deviate beyond a specified range, forces are applied to the polygonal edge nodes to attempt to bring them back into range. Figure 4-19 shows a rectangular polygon, with the desired proportion shown as a dashed rectangle, the proportion range shown as dotted rectangles, and proportion objective forces applied to the edge line nodes. The forces applied by a proportion objective are analogous to a cube of clay or Jello in that one pair of forces that squeeze the cube along one axis results in another set of forces that expand the cube along the other axes, thus preserving its volume.

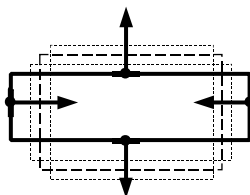


Fig. 4-19. Proportion objective

Proportion objective forces need to be defined in such a way that they change the proportion but not the area of the shape they are applied to. Figure 4-20 shows a space's polygonal shape with current proportion P and area A and its intended shape with target proportion P' and the same area. The point \bar{p}_c is the position of

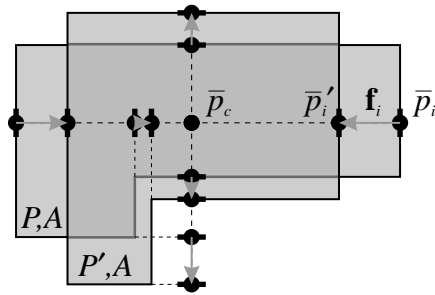


Fig. 4-20. Proportion objective forces

the space's center node, \bar{p}_i is the current position of one of the edge nodes of the polygon defining the space, \bar{p}'_i is the target position of the same node, and \mathbf{f}_i is the force vector needed to move \bar{p}_i toward \bar{p}'_i . Scaling a polygon non-uniformly changes its proportion. The correct set of horizontal and vertical scale factors can change a polygon's proportion without changing its area. The proportion of a polygon is defined here as the ratio of the horizontal distance to the vertical distance of its bounding box

$$P = \frac{\delta x}{\delta y}.$$

The horizontal scale value necessary to scale a polygon with proportion P to proportion P' is

$$s_x = \sqrt{\frac{P'}{P}},$$

and the vertical scale is then the inverse of the horizontal scale

$$s_y = \frac{1}{s_x}.$$

Each point \bar{p}'_i is found by scaling the horizontal distance between \bar{p}_i and \bar{p}_c by s_x and

the vertical distance by s_x

$$\bar{p}'_i \cdot x = \bar{p}_c \cdot x + s_x \bar{p}_i \cdot x,$$

$$\bar{p}'_i \cdot y = \bar{p}_c \cdot y + s_y \bar{p}_i \cdot y.$$

Since the position of a space's wall nodes are stored in the space's local coordinate system with its center node at the origin, as described in section 4.1.3.1, \bar{p}_c is the origin and the components of \bar{p}'_i become

$$\bar{p}'_i \cdot x = s_x \bar{p}_i \cdot x,$$

$$\bar{p}'_i \cdot y = s_y \bar{p}_i \cdot y.$$

The configuration error between \bar{p}_i and \bar{p}'_i for each edge node is then

$$\mathbf{e}_i \cdot x = (s_x - 1) \bar{p}_i$$

$$\mathbf{e}_i \cdot y = (s_y - 1) \bar{p}_i.$$

As proportion P approaches P' , s_x and s_y approach 1, and each of the errors \mathbf{e}_i approach zero.

The proportion objective configuration force for each node i is

$$\mathbf{F}_{pr\ i} = k_{pr} I_{pr} \mathbf{e}_i.$$

For the line node polygon representation used here, this method for determining proportion objective forces only works for orthogonally shaped polygons. It does not work for polygons with diagonal edges, because in a non-uniform scaling of a polygon, diagonal edges change angle. In the line node representation, the angle is stored in the node as the direction vector, and does not change after a scale. This method would work for a vertex polygonal representation. It is felt that this is not

an important limitation, because a proportion objective would not be defined for irregularly shaped spaces, and they are not as important as other types of objectives such as adjacency and area.

If a proportion range is defined for the proportion objective as described above, then s_x and s_y also depend on the target proportion range. If the current proportion is too small ($P < P_{min}$), then $P' = P_{min}$, while if it is too large ($P > P_{max}$), then $P' = P_{max}$. If the current proportion is within the desired range ($P_{min} < P < P_{max}$), then $P' = P$ and $s_x = s_y = 1$, resulting in no needed change in proportion and thus no forces applied to the nodes.

4.2.6.3. Relationship between area and proportion objectives

There is a very close relationship between the area objective and the proportion objective. If one is not designed with the other in mind they will end up fighting against each other. The forces computed for the area objective need to be defined in such a way that they do not cause the shape to change proportion. Likewise, the forces computed for the proportion objective need to be defined in such a way that they do not cause the shape to change area. If this is not done and both are applied to the same space, the space will end up oscillating. In trying to achieve a desired area, the Area Objective might change the proportion as well. Then in trying to achieve a desired proportion, the proportion objective might change the area.

4.2.7. Binary geometric objectives

Binary Geometric Objectives apply forces to two wall nodes from two different spaces. The intent of a binary geometric objective is to change the locations of two edges so as to achieve a specified geometric configuration. The data structure for binary geometric objectives minimally contains two line nodes


```

class Binary Geometric Objective
  node1 : Line Node
  node2 : Line Node.

```

4.2.7.1. Alignment objective

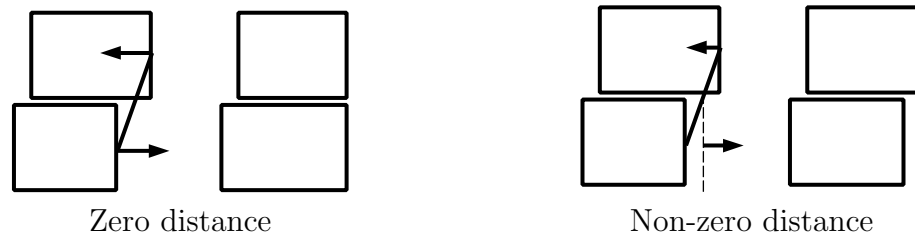


Fig. 4-21. Alignment objective

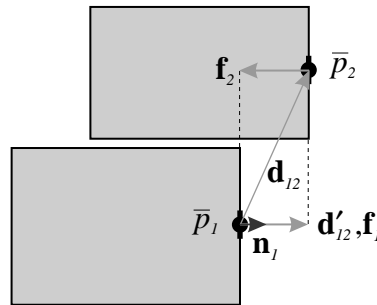


Fig. 4-22. Alignment objective forces

An alignment objective is a type of geometric objective that attempts to align two nodes or to have them separated by a specified distance. The data structure for an alignment objective contains two nodes, at least one of which must be a line node, and a separation distance. If each node is a line node, the direction vectors

of both should be parallel. Each node is connected with a spring with a signed rest length equal to the specified distance. Figure 4-21 shows the effect of an alignment objective on two sets of parallel line nodes, with both zero and non-zero distance.

Any non-trivial design problem will probably be over-defined, in that there may be more than one objective applied to the same node. In such a condition, an alignment objective represented with a simple spring and dashpot will rarely align its two nodes. For this reason an integral spring is used. The force applied by an integral spring contains the spring and dashpot components described in section 3.3.4, as well as a third component that continuously increases as long as the desired rest length is not achieved. This third component is the sum of the errors of previous time steps, where the error at a given time is the difference between the length at that time and the rest length. As the simulation time increases, this value increases until the error is zero, causing the spring to apply just enough force to make the current length equal to the rest length.

Figure 4-22 shows two rectangular spaces whose right edges need to be aligned. The vector from *node1*'s position \bar{p}_1 to *node2*'s position \bar{p}_2 is

$$\mathbf{d}_{12} = \bar{p}_2 - \bar{p}_1,$$

which is projected onto the line node normal vector of *node1* to yield

$$\mathbf{d}'_{12} = \mathbf{n}_1(\mathbf{d}_{12} \cdot \mathbf{n}_1).$$

Since \mathbf{n}_1 is by definition a unit vector, \mathbf{d}'_{12} is a vector that is perpendicular to each line node, and whose length is the distance between their parallel direction vectors.

The rest length r of the integral spring is set to the specified separation distance. It is a signed value, measured relative to the normal vector of *node1*.

By definition, the error e_0 at time step 0 is zero, where time step 0 is the first

time step when the alignment objective is to be applied. The error at time step n is

$$e_n = e_{n-1} + \mathbf{d}_{12} \cdot \mathbf{n}_1.$$

e_n continually changes in magnitude until the objective is met, at which time

$$\mathbf{d}_{12} \cdot \mathbf{n}_1 = 0.$$

The configuration error on *node1* at time t_n is

$$\mathbf{e}_1 = (|\mathbf{d}'_{12}| - r) \frac{\mathbf{d}'_{12}}{|\mathbf{d}'_{12}|} + e_n.$$

While the configuration error on *node2* is the opposite of that of *node1*

$$\mathbf{e}_2 = -\mathbf{e}_1$$

The alignment objective configuration forces for the nodes are

$$\mathbf{F}_{al1} = k_{al} I_{al} \mathbf{e}_1$$

$$\mathbf{F}_{al2} = k_{al} I_{al} \mathbf{e}_2.$$

4.2.8. Relationship between topological and geometric objectives

One of the issues that must be addressed is what happens when spaces overlap. In physically based modeling this issue is called ‘collision detection and response.’ The objects in the system need some way to tell when they have collided with each other, and some way to respond or change their motion as a result of this collision. The design of topological and geometric objectives is generally governed by the collision method used during their respective resolutions; that is, topological objectives are designed given the circular space representation used for collision detection during topological resolution, while geometric objectives are designed given the rectangu-

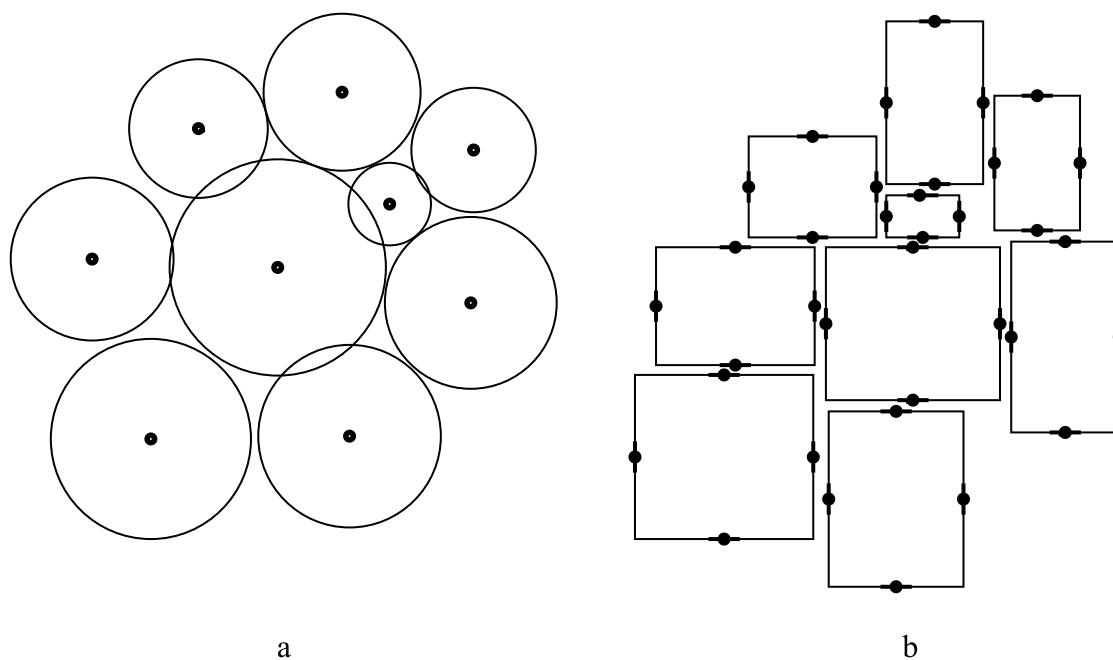


Fig. 4-23. Topological elements vs geometric elements

lar or polygonal space representation used for collision detection during geometric resolution.

Figure 4-23 shows how the same space planning problem is essentially two separate problems. 4-23a shows the topological view of the problem, with circular shaped spaces and point nodes at their centers, while 4-23b shows the geometric view of the problem, with rectangular shaped spaces and line nodes at wall midpoints. The two problems are ‘connected’ via the local coordinate system of each space (see section 4.1.3.1). Wall node positions are defined within their space’s origin, which is the space center node. They are then transformed into world coordinates so that the dynamical system can work on them.

One consequence of this dichotomy is that although a topological objective might be satisfied during topological resolution, after switching to geometric resolution it

might no longer be met. For example, say that two rectangular spaces need to be adjacent to each other. Although their circular representations might be adjacent during topological resolution, they rarely will be during geometric resolution. For this reason, and to make this approach more ‘robust,’ topological objectives should really be implemented in two different ways, one for circular collisions and the other for polygonal collisions.

4.3. Process

Once a set of spaces and objectives has been defined, a dynamic simulation runs through a series of phases to produce a layout solution. First, topological relationships between spaces are resolved. Second, the geometric positions of walls separating the spaces are resolved. Finally, the designer interacts with the design by modifying the set of design objectives, thereby modifying the design itself. This is conceptually a linear process, but in reality it is highly circular. Every modification of design objectives causes the simulation to loop to either topological resolution or geometric resolution, depending on the type of design objective being modified. This process is shown graphically in figure 4-24.

4.3.1. *Topological resolution*

The first phase in solving a space layout is to determine the location of each space relative to all other spaces. In this phase only topological objectives are applied. For collision detection, boundary shapes are treated as circles so spaces are able to slide around each other. If polygonal boundary shapes were used, corners might catch on each other and keep one space from being able to move to the other side of another. The dynamic simulation runs until the system is in equilibrium, which is defined as

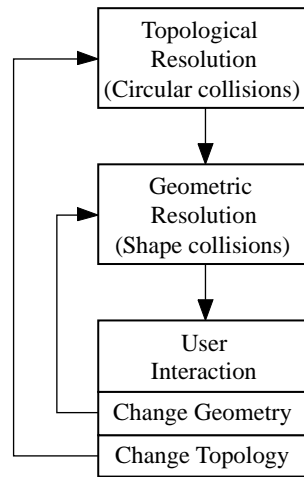


Fig. 4-24. Simulation

the state at which there are no unbalanced forces acting on a body.

4.3.2. *Geometric resolution*

Once the topological simulation has reached dynamic equilibrium, as described in section 3.3.9, the second phase is started, during which geometric objectives are applied and topological objectives are turned off. In this phase, space boundaries are switched from a circular to a polygonal representation. Collision detection and response, as described in section 3.3.6, then act to keep spaces from overlapping, resulting in an arrangement that is very close to a recognizable building floor plan. Constrained dynamics is used to maintain the separation between spaces, which are more fully described in section 3.3.7.

4.3.3. *User interaction*

At any time while the simulation is running, but typically once a geometric simulation has reached equilibrium, the designer can analyze and interact with the design by

modifying existing objectives and adding new ones. Here is where the true power of this approach becomes apparent. The designer interactively manipulates the design via objectives rather than via geometry, allowing him or her to concentrate on the design itself rather than on the mechanics of geometric transformations.

4.3.4. Topological/geometric connection

The topological and geometric systems appear to be totally disconnected systems in that topological objectives only apply forces to space center nodes and never to polygon edge nodes and conversely geometric objectives only apply forces to polygon edge nodes and never to space center nodes. The key to the connection between the two lies in each space's local coordinate system (see section 4.1.3.1). The edge node positions of each polygon are defined in its space's local coordinates. Any force that moves an edge node indirectly moves the space's center node, and any force that moves the center node moves all edge nodes.

4.4. Summary

I have described the components of a physically based space planning system, which include design elements such as nodes, polygons, and spaces that are used to define the tangible parts of a design, and a number of design objectives used to define the intangible intent of the designer. Two types of design objectives were defined: 1) topological objectives that affect the location of one space relative to another, and 2) geometric objectives that affect the dimensional size of individual spaces. Finally, a procedure was defined for using these elements and objectives in a design process.

To generalize to any design domain dealing with multi-dimensional space, design elements and design objectives need to be defined for each domain, while the

algorithms should apply to every domain. Design elements are the ‘objects’ that the designer is working on, while design objectives are the intentions of the designer to change the state of those objects. The concepts of elements and objectives apply to every design domain, while the specific elements and objectives described here apply only to the domain of space planning, and may be slightly modified in terms of nomenclature to apply to general floor planning problems. The algorithms described here define the means by which design objectives operate on design elements, and should generally apply to every design domain. Conceivably then, given a generalized physically based design methodology, for every design domain it should be possible to define a unique set of elements and objectives specific to that domain, while the algorithm remains the same across domains.

5. PROTOTYPE IMPLEMENTATION

This section describes the implementation of a prototype software application that demonstrates the concepts of the previous chapter.

5.1. Implementation

Here I give some of the details of the prototype implementation, and discusses some of the significant differences between how concepts were described in section 4, “A physically based approach to space planning,” and how they were implemented in the prototype. Appendix A provides a detailed description of the interface and how to use it. A general discussion of this implementation is left for the next section.

5.1.1. *Implementation summary*

Table 5-1 shows all the main concepts presented in section 4, “A physically based approach to space planning,” along with an indication of whether or not and to what degree they were implemented in the prototype.

As can be inferred from the table, spaces are limited to rectangles in a single story building. Limiting spaces to rectangular shapes has often been used in prototypes to demonstrate new approaches (Flemming, 1978). If an approach works for rectangles, chances are it will work for other shapes. The problems of circulation and multi-story hierarchical design are left for future investigation.

5.1.2. *Polygons*

The line node polygonal representation described in section 4.1.2 was implemented in the prototype. In addition to the advantages described in that section – easy computation of forces to maintain edge orientation – it allowed a relatively sim-

Table 5-1. Implemented concepts from section 4

Item	Implemented
Nodes	
Point Node	Yes
Line Node	Yes
Plane Node	No
Polygons	
Rectangular edge list	Yes (see section 5.1.2)
Non-rectangular edge list	No
Non-rectangular vertex list	No
Space	
Space Object	Yes
Space Hierarchy	No (see section 5.1.3)
Unary Topological Objectives	
Interior	Yes
Exterior	Yes
Orientation	Yes
Binary Topological Objectives	
Adjacency	Yes
Separation	Yes
Unary Geometric Objectives	
Area	Yes
Proportion	Yes
Binary Geometric Objectives	
Alignment	Yes
Automated Simulation	Yes

ple implementation of physically based techniques using rectangles. The polygon polarities discussed were not implemented.

5.1.3. Space hierarchy

Section 4.1.3 discusses a multi-level spatial hierarchy, but in the prototype only one hierarchical level was fully implemented. The prototype space code was structured to implement a multi-level hierarchy, as can be seen in the structure of an .apf file (see appendix B). However, the prototype dynamical system was not structured to account for multi-level hierarchy.

5.1.4. Details

This prototype was initially developed on a Silicon Graphics O2 workstation using the Irix 6.3 operating system, with a 200 MHz R5000 processor, and 192 MB RAM. Programming was done in object-oriented C++, using OpenGL for the graphics, and the Fast Light Tool Kit (FLTK) user interface toolkit (FLTK, 2003), version 1.1.4rc1. The prototype was later ported to the Microsoft Windows 98 and 2000 operating systems using Microsoft Visual C++ versions 6.0 and 7.0, and further developed on 650 MHz and 1200 MHz Intel Pentium III processors. Additional software libraries include gltt version 2.5.2 (GLTT, 2001) used to draw True Type text in OpenGL applications, and libpng version 1.2.5 (PNG, 2002) used to create png (Portable Network Graphic) images.

5.2. Suggested interaction

Here I describe a suggested process for interacting with and using the *Physically Based Space Planning* prototype. This seemed necessary, because like all prototype software it has designed shortcuts that make user interaction uncertain. Appendix A gives detailed and complete information on the user interface.

Labels mentioned in this section are displayed with *italics*.

Run *pbspace.exe*. When the application is displayed an empty space plan consisting of a top-level project space with a single space node is drawn in the drawing area.

Create some spaces. Using the left mouse button, create a number of rectangular spaces.

Define some adjacency objectives. Using the right mouse button, create a number of adjacency objectives between spaces by clicking on one space node and

dragging to another.

Define additional topological objectives. Add some additional design objectives by selecting them from the *Objectives* menu and following the directions of the active help displayed below the drawing area.

Set *Intersections to Circles* and turn off *Auto*. To the left of the drawing area, set the intersection method to *Circles* and turn off *Auto*, which automatically switches the method from circles to polygons when the simulation reaches dynamic equilibrium. It is useful in the beginning to do this manually so as to better understand the differences between them.

Run the simulation. Click the *Start* button to the left of the drawing area. The spaces will contract toward each other and come to rest within a few seconds, or longer if one space is in the process of moving around another. The *Start* button now reads *Stop*. If at any time the plan becomes unstable or moves off the screen click the *Stop*, *Reset*, or *Reset Random* buttons.

At this point the dynamical space plan is trying to achieve topological resolution. The rectangular spaces will overlap because as far as the system is concerned at this stage they have circular shapes. Click on the *Circle* and *Polygon* buttons under the *Display* tab to display the spaces as circles instead of rectangles. The circles may overlap a small amount, but for the most part they do not.

Reposition spaces. Click and drag on the center node of a space to reposition it. Notice how the other spaces reposition themselves in response. (It is possible at this time that an external force is introduced due to accumulated error that might eventually move the plan off the screen. Either *Reset*, or use the middle mouse button to move the plan toward the center.)

A useful process for repositioning spaces is to position them based on the relative importance of their adjacency objectives. Go to the *Importance* tab and turn

off the display of all but *Mandatory* importances. Reposition spaces until these adjacencies are met. Turn on the display of *Significant* importances and do the same for them. This process, performed manually here, could be automated, as described in section 4.3.1.

If spaces are drawn as circles at this point, click on the *Circle* and *Polygon* buttons under the *Display* tab again to display the spaces as rectangles.

Switch to geometric resolution. Once the system has reached equilibrium, click on the *Polygons intersections* button to the left of the drawing area, which also turns off topological objectives and turns on geometric objectives. The dynamical space plan is now trying to achieve geometric resolution, and you should notice that the rectangular spaces no longer overlap and that the space plan expands by a small amount because the adjacency objectives are no longer active. The space plan should now begin to look more like a potential building plan.

Define geometric objectives. From the *Objectives* menu, select *Alignment*. Locate two walls that you want to be co-linear, click and hold on one with the right mouse button, drag to and release the button over the other. The two walls should begin to align. Do this with a few other walls.

The process of adding alignment objectives begins to reveal the power of this approach to space planning. During topological resolution, because the spaces overlap each other, few space arrangements appear to be that of a recognizable building. Once the switch to geometric resolution is made, spaces no longer overlap and the arrangement *begins* to look more recognizable, but usually side gaps still exist between spaces and the layout looks haphazard and not intentionally ‘designed.’ Once alignment objectives or other geometric objectives are added to the plan, the designer’s aesthetic intent is inserted, and the plan begins to look more recognizable as a building, thus revealing physically based space planning as a compelling design

method.

The essence of the physically based space planning approach is demonstrated with the steps outlined so far. The following steps reveal other information and the possibilities of their use.

Display area error Under the *Options* menu, turn off *Black and White* to display colored graphics on a black background. From the *Display* tab, click on the *Shaded Area* button. Some of the spaces will now be colored shades of red or blue. Darker red areas are smaller than the required area specified in the architectural programming file, while darker blue areas are larger; the darker the color, the more the error.

Display force vectors From the *Display* tab, click on the *Vectors* toggle, then click on the *Step* button to the left of the drawing area. All individual force vectors active at the current state of the dynamics system will be drawn. See sections A.4 and A.4.3 for explanations on how to control the display of the different force types.

5.3. Early worked example

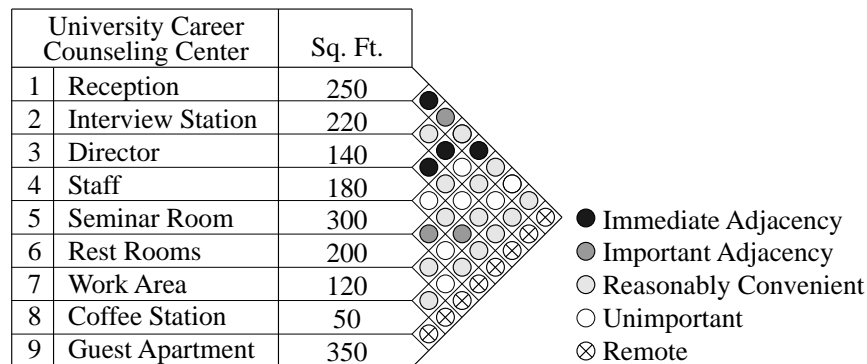


Fig. 5-1. Sample Adjacency Matrix [Redrawn from Karlen (1993, p. 22)]

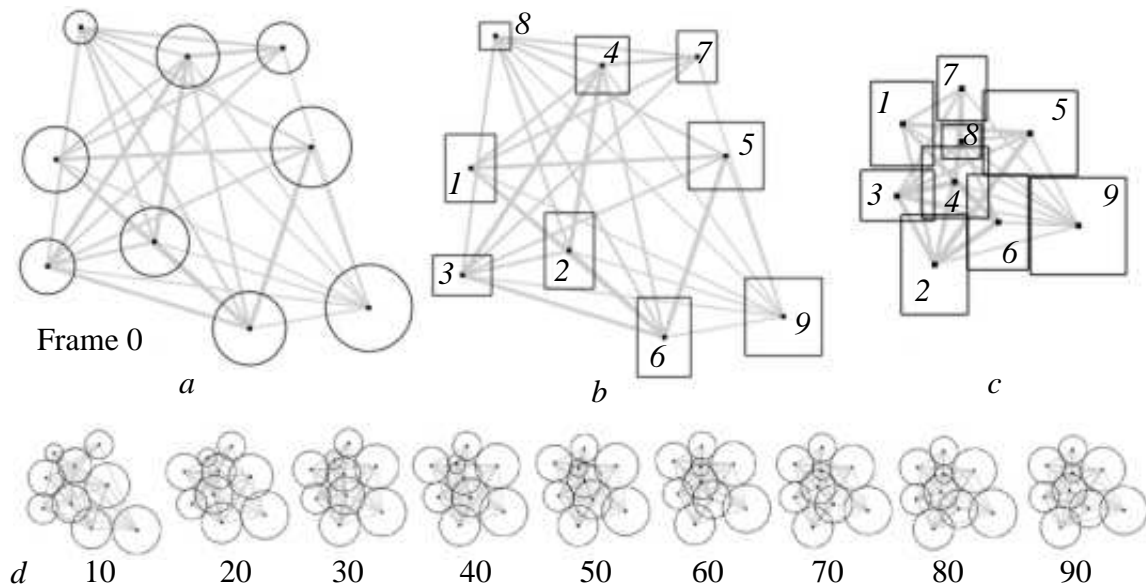


Fig. 5-2. Sample topological resolution

During early development of the prototype, the architectural program shown in figure 5-1 was used to define the initial space planning requirements. During the early stages of development, the architectural program needed to be fairly small but contain many adjacency requirements. It needed to be small because the emphasis in early development was in getting the dynamics between individual spaces to work, not in working on a large space planning problem. It needed to have many adjacency requirements so that many locally optimal solutions were possible. A small number of adjacency requirements would yield a small number of solutions.

Figure 5-2 shows a sample topological resolution using the architectural program in figure 5-1. The APF file used to create these figures was similar to that listed in appendix C. Figure 5-2a and 5-2b show each space boundary drawn with its required area and with random initial positions, figure 5-2a displaying boundaries drawn as circles, and figure 5-2b displaying boundaries drawn as rectangles with

random proportions. Recall that during topological resolution, circles are used in collision detection. It is difficult to show the dynamic movement with static images, but figure 5-2d shows every tenth frame from the dynamic simulation, with frame 90 showing the spaces in equilibrium. The entire sequence took about three seconds to compute and display, so the illusion to the user is of smooth natural motion. Notice that most of the movement occurs between frames 0 and 10 when the spaces are coming together, and that any movement after that is a result of the spaces rearranging themselves and coming to equilibrium. With some initial positions it is possible for the system to almost be at equilibrium when one space manages to move onto the other side of another, and the whole system rearranges itself. Figure 5-2c shows the final topological solution with the boundaries drawn as rectangles again. Although some of the boundaries overlap, this is not important during topological resolution and the overlaps will be resolved during geometric resolution.

Figure 5-3 shows six samples of geometric resolutions using the same architectural program. Initial proportions for each space were maintained from sample to sample, but initial positions were randomized. Final topological relationships were not edited manually. For these results, topological resolution was not performed before geometric resolution, so these do not represent locally optimal topological solutions. Note in figure 5-3f that some wasted space is possible.

Note the variety of designs produced from a simple set of objectives. The only objectives active in producing these samples are adjacency and rectangular area objectives. The addition of other objectives such as non-rectangular shape, parental shape, and alignment objectives, among others, should allow the architect to have a great amount of control over the design of space plans.

In another worked example, figure 5-4 shows two more results from a sample design problem using the same architectural program described above. Figures 5-4a

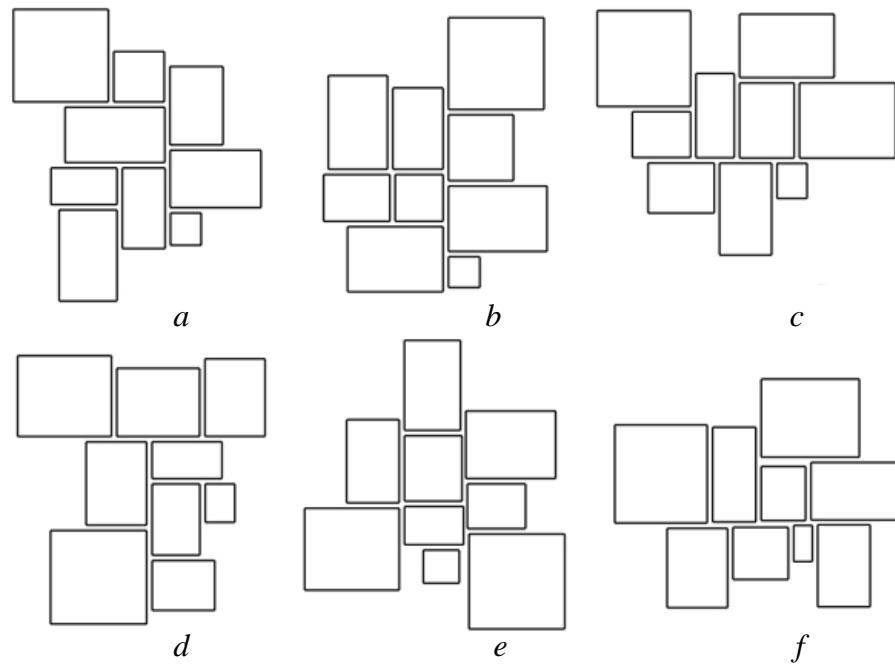


Fig. 5-3. Sample geometric resolutions

and 5-4b show the same set of spaces with the same initial random positions, but with a different set of topological objectives. Adjacency objectives are solid lines connecting spaces, the width of the line representing the strength of the adjacency. Separation objectives are dashed lines connecting spaces, in this example all with the same separation strength. Figure 5-4b shows the same problem as in 5-4a, but with the addition of an interior objective for space 3, an exterior objective for space 6, and an orientation objective for space 7.

Figures 5-4c through 5-4g show the process of resolving a space plan from the initial state shown in 5-4b. 5-4c shows the result of topological resolution after the simulation has reached equilibrium. Recall that during topological resolution space boundaries are treated as circles for collision detection purposes. Note the location of space 9, which is connected to all other spaces with separation objectives. This

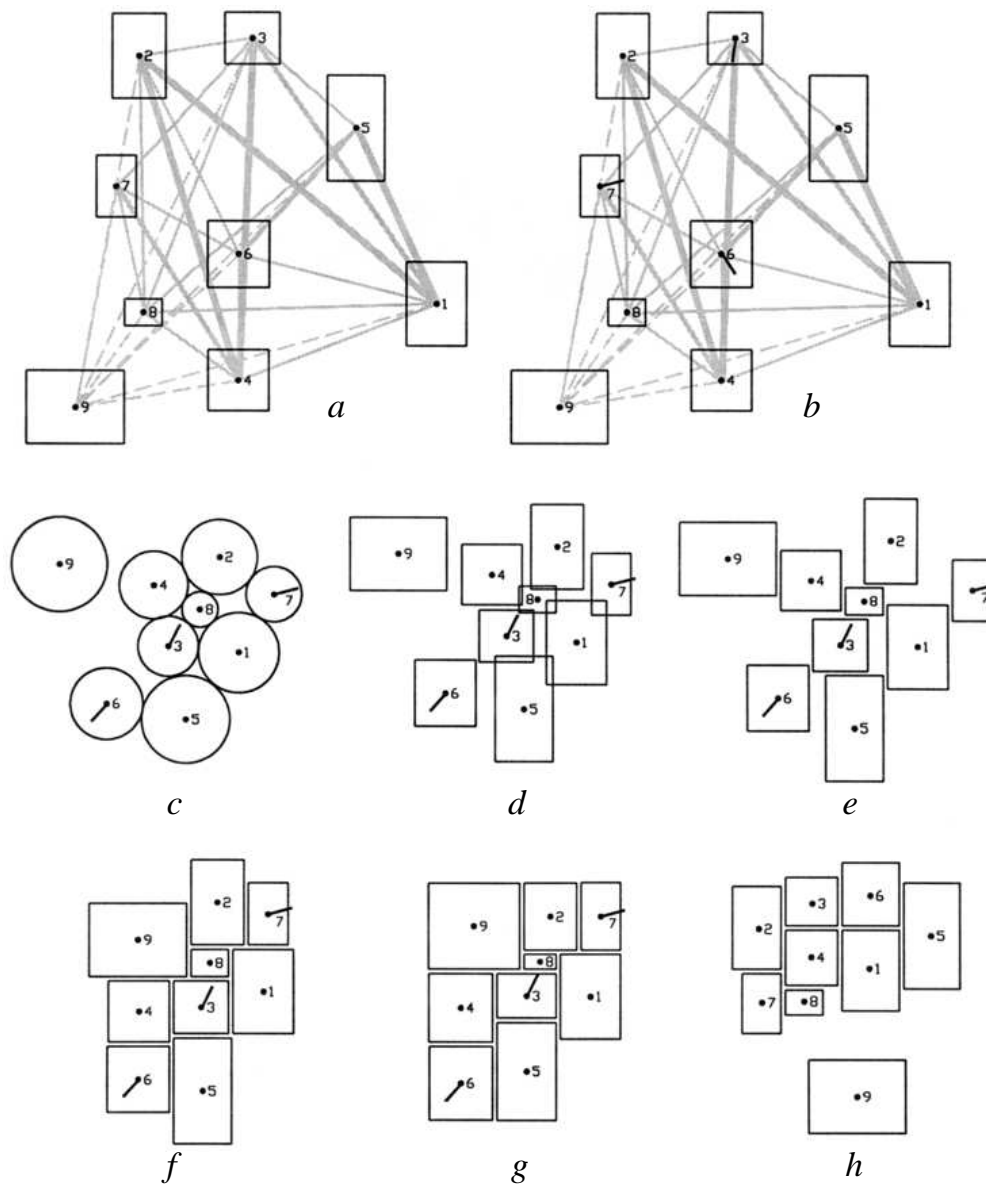


Fig. 5-4. Early worked example

physical separation demonstrates the problem of separation objectives described in section 4.2.5.2. 5-4d shows the transition of space boundary representation from circles to polygonal shapes, and shows the many gaps and overlaps between shapes. 5-4e shows the first step in geometric resolution, with all topological objectives turned

Table 5-2. Dynamic and other constants

Coordinate system units	feet
Wall thickness	1 foot
Drawing area screen size	100' x 100'
Time Step	1/30 second (30 frames per second)
Adjacency spring constant	0.0 - 20.0
Shape spring constant	500.0
Edit spring constant	500.0
Spring dashpot	2.0
Coefficient of Restitution	0.0
Viscosity	10.0
Mass	1.0

off, and geometric objectives except gravity turned on. The overlaps are removed, but the gaps remain. 5-4f shows the result of manually moving spaces until they contact each other (see section 4.3.2), which removes the gaps. This figure also shows the beginnings of manual design interaction with the minor relocation of spaces 4, 7, and 9. Finally, 5-4g shows more designer manipulation with the addition of alignment objectives to clean up the outer walls.

Figure 5-4h shows one step near the end of the process of resolving a space plan from the other initial state shown in 5-4a. It is at a similar state in the process as figure 5-4f, but without any manual space relocations. Recall that 5-4a and 5-4b have the same set of adjacency objectives. Note how the addition of interior, exterior, and direction objectives on spaces 3, 6, and 7, respectively, affect their final locations in figure 5-4f.

Table 5-2 shows some of the mathematical values used in the physically based simulation and in the drawing area to create the images shown in this section. They apply to the physically based simulation and not to the design problem used in the worked examples, and were found to provide acceptable behavior in the simulation and can be used as a starting point for further investigation and refinement.

5.4. Final worked example

The images shown in the previous section were created during early development of the prototype. Later development included refinement of most objectives, that is, interior, exterior, orientation, separation, area, and proportion objectives, development of offset alignment objectives, use of a local coordinate system in spaces for wall nodes, extension of level of importance from adjacency objectives only to all objectives, large amount of user interaction work, and implementation of automatic switching from topological to geometric resolution.

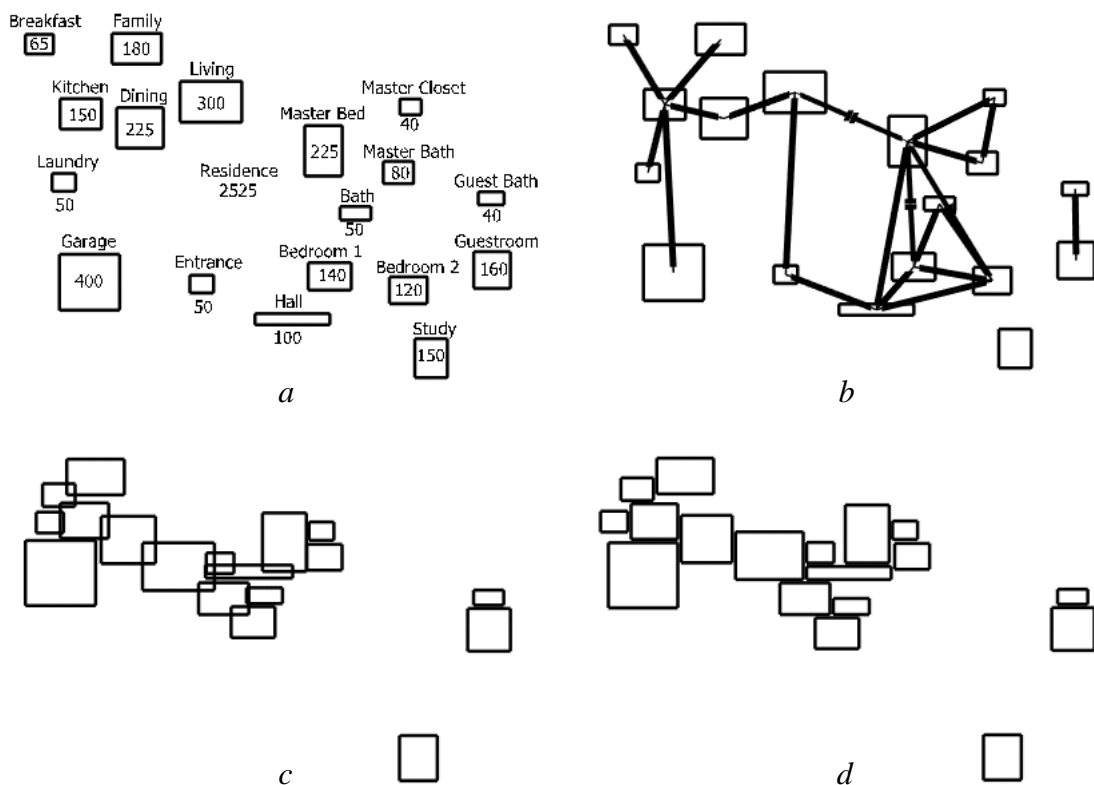


Fig. 5-5. Final worked example - from initial placed positions

Figures 5-5, 5-6, and 5-7 show space plans at varying stages of the resolution process for a residential program using the later prototype. The APF file used to create these figures was similar to that listed in appendix D.

Figure 5-5 shows four figures at the early stage in the design process, where the spaces in the architectural program were initially created in a rough relationship to each other. In figures 5-5a and b the spaces are at their initial positions, figure a displaying space names and areas, and figure b displaying all design objectives. Figure 5-5c shows the result of topological resolution from the initial state in the previous figures, while figure 5-5d shows the continuing result of geometric resolution.

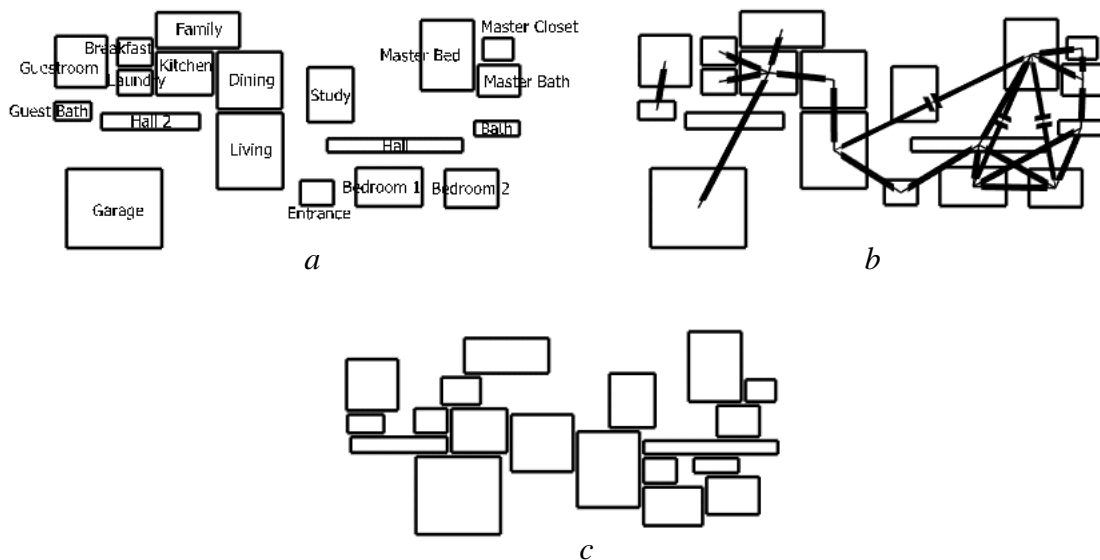


Fig. 5-6. Final worked example - after manual rearrangement of spaces, and doing topological and geometric resolutions

Figure 5-6 shows three figures at the next stage of design, where a few spaces are manually rearranged, especially those without any adjacency objectives, figure 5-6a displaying names and figure b displaying design objectives. Figure 5-6c is similar to

figure 5-5d, showing the result of topological then geometric resolution.

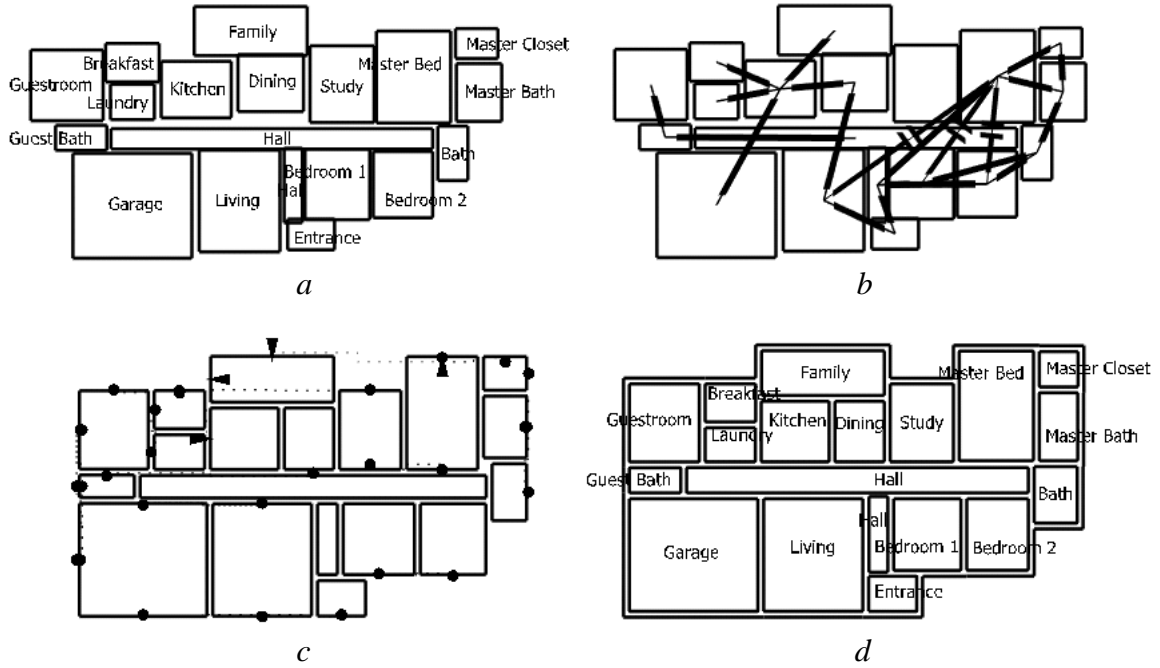


Fig. 5-7. Final worked example - after manual rearrangement of spaces, and doing only geometric resolution

Figure 5-7 shows four figures much later in the design process, after much manual manipulation occurred after topological resolution but before final geometric resolution. Again, figure 5-7a displays space names and figure 5-7b displays design objectives. Figure 5-7c shows the result of geometric resolution, after the many alignment objectives shown were added, and figure 5-7d shows the final result with space names along with exterior wall surfaces.

5.5. Summary

The vast majority of the concepts described in section 4, “A physically based approach to space planning,” were successfully implemented in the prototype application. The prototype adequately demonstrates the intended concepts and suggests that this approach is a potentially promising computer aided design methodology. The prototype also pointed out some potentially severe limitations, such as performance problems, that must be overcome and additional features, such as a contact objective, that must be designed and implemented.

The implementation of the prototype was an extremely valuable process in the act of defining the concepts described in section 4, and the two, concepts and prototype, were developed in iterative stages. Many fully defined concepts, once implemented, were determined to be ineffective in some way. Or, many new concepts were formed based on interaction with the implementation. The implementation, then, informed and modified those concepts to their current state, and was an integral part of the process.

Now that the prototype implementation has been described, along with the results of a small worked example, the next section presents a computational and space complexity analysis, which will provide an indication of how this approach will scale to larger space planning problems, and will also identify those processes where performance improvements or alternate implementations will have the greatest impact.

6. COMPLEXITY ANALYSIS

Space and computational complexity will be discussed using the $O()$, or ‘big O , notation typically used in computer science, which provides an asymptotic analysis for comparing different algorithms (Russell and Norvig, 1995, p. 851).

In the following discussion, n is the number of nodes, s is the number of spaces, d is the number of design objectives, c is the number of contacts, or pairs of overlapping spaces, and k is the number of edge or wall intersections. At times when the task performed for each space is a function of the number of nodes in the space, $s = n$.

As discussed further below a worst case scenario involves an arrangement of polygons with a very large number of intersections. It is possible to artificially construct such an arrangement, but in typical practice architectural floor plans are more “well behaved” and do not contain a large number of intersections for each space. The results below show that the worst case computational complexity is $O(n + d + n^2 + n^2c + s(n + k) \log n)$, while the expected case computational complexity is $O(n^2)$, which is due to the task of computing dynamic constraints. Further, the worst case space complexity is $O(n^2)$, while the expected case space complexity is $O(n)$.

6.1. Computational complexity

The overall computation complexity of the algorithm is determined by the complexity of one time step in the physical simulation, which is called the DOTIMESTEP algorithm and shown later in figure 6-5. I will first discuss the computational complexity of distinct parts of this algorithm, put them all together, and then reduce them to arrive at a measure of the overall computation complexity.

In this section, algorithms will be described in figures using lines of pseudo-code

on the left, and $O()$ notation for the computational complexity of the line on the right. A line of pseudo-code and its corresponding $O()$ notation will be indented to the right of the line above it if it is contained within a **for** loop or an **if** condition. In this way, the complexity of a line of pseudo-code is easily seen as the sum of the complexities of indented lines below it.

6.1.1. Reductions

Since many design objectives apply to more than one node, it is conceivable that each new node added to a system adds design objectives for all other nodes, which means that $d \approx n^2$. However, even highly complex architectural floor plans will not require this many objectives. For example, a room in a building with a thousand rooms will not have an adjacency objective with most of the other nine hundred ninety-nine. In practice d increases in size at a more constant rate for each additional node, making $O(d) = O(n)$. This supposition will be used to reduce and simplify the computational complexity analysis in the following discussion, which will be indicated with the \Rightarrow operator. The \Rightarrow operator will also be used to indicate that alternative implementation methods exist that can further reduce the complexity of various tasks.

Another area of reduction that is possible after some analysis is the number of contacts c and the number of edge intersections k . Given a number of spaces with a total of n edges, it is possible to artificially construct a placement such that each space overlaps all others, in which case the size of both c and k approaches n^2 . However, due to collision detection and response and the maintaining of space separations using dynamic constraints, it is *not* possible for each space to overlap every other space. So k is limited by the number of space-to-space contacts that are possible. Most of the spaces in a typical architectural space plan are of relatively

similar size, the larger spaces being on the order of 10 times larger than the smaller spaces, with the exception of very large gathering spaces like stadiums and convention centers where this ratio may be more on the order of 100 or even 1000. If we can also discount the number of spaces in contact with long, thin spaces such as corridors, then c and k can be further reduced to a constant value for each node, or $O(c) = O(n)$ and $O(k) = O(n)$.

6.1.2. INTEGRATE Algorithm

Algorithm INTEGRATE (N, D) $O(n + d + n^2) = O(d + n^2)$
 $\Rightarrow O(n^2)$

Input:

- N , an unordered list of n nodes contained in S
- D , an unordered list of d objectives

Output:

- N , with new position and velocity for each node

- | | |
|---|----------------------------|
| 1. for n nodes in N | $O(n)$ |
| 2. Set initial environmental forces | $O(1)$ |
| 3. for d objectives in D | $O(d)$ |
| 4. Apply objective forces | $O(1)$ |
| 5. Compute dynamic constraints | $O(nc) \Rightarrow O(n^2)$ |
| 6. for n nodes in N | $O(n)$ |
| 7. Differentiate | $O(1)$ |

Fig. 6-1. Simplified algorithm to integrate over a set of nodes

Figure 6-1 shows the pseudo-code for the INTEGRATE algorithm. Lines 1, 3,

and 6 are constant time tasks for each node and objective, and together have a complexity of $O(n + d)$. Line 5 computes dynamic constraints, which are described in sections 3.3.7 and 4.3.2. See Witkin and Baraff (1997, p. F1–F12) for a complete description of the required implementation, which involves the use of a biconjugate gradient solver (Press et al. (1992, p. 83)). Solving dynamic constraints involves inversion of an n by n matrix, but by using sparse matrix techniques the complexity of the biconjugate gradient solver is actually $O(nc)$. For worst-case systems with many collisions between nodes this approaches $O(n^3)$, whereas for expected-case architectural floor plan systems this approaches $O(n^2)$.

The overall computational complexity of the INTEGRATE algorithm is $O(n + d + n^2)$, which equals $O(d + n^2)$, and if $O(d) = O(n)$ as discussed above is further reduced to $O(n^2)$ in practice.

6.1.3. HANDLECONTACTS Algorithm

Figure 6-2 shows the pseudo-code for the HANDLECONTACTS algorithm. Line 1 is a constant time task for each contact to remove those between spaces that have moved apart, and line 8 is a constant time task for each contact to set values. Each has a complexity of $O(c)$.

The purpose of the task in line 3 is to find contacts between spaces. The prototype implementation used a brute force method of iterating through each space pair, searching the list of contacts to see if the contact already exists for that pair (lines 4 and 5), testing if they are in contact (line 6), and inserting the contact in the list if needed (line 7). Iterating through the space pairs in line 3 has complexity $O(s^2)$, but since the contact test is performed for each node in the spaces boundary, the actual complexity is $O(n^2)$. As implemented, line 4 has a complexity of $O(c)$, but existing search techniques make it possible to achieve $O(\log c)$. So the total complexity of

Algorithm HANDLECONTACTS (S, C) $O(c + n^2c) = O(n^2c)$
 $\Rightarrow O(n(\log n)^2)$

Input:

S , an unordered list of s spaces
 C , an ordered list of c contacts

Output:

C , a new list of contacts

1. for c contacts in C	$O(c)$
2. Remove if not valid	$O(1)$
3. for $s(s - 1)/2$ space pairs in S	$O(n^2c) \Rightarrow O(n \log n \log c)$
4. for c contacts in C	$O(c) \Rightarrow O(\log c)$
5. Search for existing contact	$O(1)$
6. if not in C and spaces are in contact	$O(1)$
7. Insert new contact into C	$O(\log c)$
8. for c contacts in C	$O(c)$
9. Post process	$O(1)$

Fig. 6-2. Simplified algorithm to find and handle contacts between spaces

lines 3-7 as implemented is $O(n^2(c + \log c))$, which equals $O(n^2c)$. With changes to use existing techniques that are more efficient, this can be reduced to $O(n \log n \log c)$.

The overall computational complexity of the HANDLECONTACTS algorithm is $O(n^2c)$, which in practice can be reduced to $O(c + n \log n \log c)$, and further reduced to $O(n(\log n)^2)$ if $O(c) = O(n)$.

6.1.4. POLYGONUNION Algorithm

Figure 6-3 shows the pseudo-code for the POLYGONUNION algorithm. Line 2 is used to set up each space in preparation of its use in the polygon union algorithm. Its

Algorithm POLYGONUNION (S) $O(n + s(n + k) \log n)$
 $= O(s(n + k) \log n)$
 $\Rightarrow O(n \log n)$

Input:

S , an unordered list of s spaces

Local:

k : count of edge intersections

Output:

For each space in S that contains child spaces, a set of polygons that is the union of the child polygons

1. **for** s spaces in S $O(n)$
2. Setup space $O(n \text{ amortized})$
3. **for** s spaces in S $O(s(n + k) \log n)$
 $\Rightarrow O((n + k) \log n)$
 $\Rightarrow O(n \log n)$
4. **if** s contains subspaces $O(1)$
5. Compute Polygon Union (recursive) $O((n + k) \log n)$

Fig. 6-3. Simplified algorithm to compute union of space polygons

running time depends on the number of nodes in the space, so its complexity is $O(n)$. The complexity of line 1 would then be $O(sn)$, except that the task in line 2 is only performed once for each node, so it is amortized across all the nodes, and the resulting complexity of line 1 is $O(n)$.

Line 5 computes the union of the polygonal outline of a space's set of subspaces, and its complexity is $O((n + k) \log n)$ as described in section E.5. It is run for each space, and is potentially recursive in that it may need to be run on each subspace that contains its own subspaces. In the worst case arrangement of a deep space

hierarchy, where each space contains at most two subspaces, each of which may or may not contain subspaces, the complexity of line 3 will be $O(s(n+k)\log n)$.

Worst case arrangements are highly unlikely in typical architectural floor planning problems. Even a highly complex building such as a hospital probably contains a space hierarchy of limited depth, and each non-leaf node in the hierarchy contains a relatively large number of leaf nodes. So, typical practice can reduce the complexity of line 3 to $O((n+k)\log n)$.

We can make a further reduction in complexity by analyzing k , the number of edge intersections. For the computation of a single space's polygon union, the worst case arrangement of its subspaces, in which each subspace overlaps every other subspace, results in k approaching n^2 . However, due to collision detection and response and the maintaining of space separations using dynamic constraints, it is impossible for each subspace to overlap every other subspace. So k is limited to the number of space-to-space contacts that are possible. Most of the spaces in a typical architectural space plan are of relatively similar size, the larger spaces being on the order of 10 times larger than the smaller spaces, with the exception of very large gathering spaces like stadiums and convention centers where this difference is more on the order of 100 or even 1000. If we can also discount the number of spaces in contact with long, thin spaces such as corridors, then k can be further reduced to a constant. So, typical practice can further reduce the complexity of line 3 to $O(n\log n)$.

The overall worst case computational complexity of the POLYGONUNION algorithm is $O(n + s(n+k)\log n)$, which equals $O(s(n+k)\log n)$, which in typical practice can be reduced to $O(n\log n)$.

Algorithm DOTIMESTEP (S, D, C) $O(d + n^2c + s(n + k) \log n)$
 $\Rightarrow O(n^2)$

Input:

S : an unordered list of s spaces
 D : an unordered list of d objectives
 C : an ordered list of c contacts

Local:

N : an unordered list of n nodes contained in S

Output:

N : nodes with new positions and velocities
 C : modified list of contacts

1. for d objectives in D	$O(d)$
2. Setup objective	$O(1)$
3. for n nodes in N	$O(n)$
4. Setup node for integration	$O(1)$
5. INTEGRATE (N, D)	$O(d + n^2) \Rightarrow O(n^2)$
6. for n nodes in N	$O(n)$
7. Post process after integration	$O(1)$
8. HANDLECONTACTS (S, C)	$O(n^2c) \Rightarrow O(n(\log n)^2)$
9. POLYGONUNION (S)	$O(s(n + k) \log n) \Rightarrow O(n \log n)$
10. COMPUTEENERGY (N, D)	$O(n + d) \Rightarrow O(n)$

Fig. 6-5. Algorithm to perform one time step

pending on the integration method used. For example, when using Runge-Kutta 4th order numerical integration it will be run 4 times for each time step. However, this does not change its complexity because it is a constant.

The overall computational complexity of the DOTIMESTEP algorithm is $O(n + d + n^2 + n^2c + s(n + k) \log n)$, which equals $O(d + n^2c + s(n + k) \log n)$ after

removing n and n^2 , which are dominated by n^2c . With a variety of reductions, either through implementation or an argument from practical applications, the expected overall computational complexity is $O(n + n \log n + n(\log n)^2 + n^2)$, which equals $O(n^2)$ after removing n , $n \log n$, and $n(\log n)^2$, which it dominates.

6.2. Space complexity

All data structures required by the algorithm, except one, are of linear complexity. That exception is the storage required to compute dynamic constraints in line 5 of the INTEGRATE algorithm shown in figure 6-1. The biconjugate gradient solver involves inversion of a n by n matrix, which would normally result in a space complexity of $O(n^2)$. However, through the use of sparse matrix techniques mentioned in section 6.1.2, a full n by n matrix is not required, resulting in an actual space complexity for this task of $O(n)$, which will not affect the overall space complexity.

Each node, space, design objective, contact, and intersection are of constant size, so the space complexity of these elements is $O(n + s + d + c + k)$. The worst case relationships between these variables, as discussed in section 6.1.1, are $O(s) = O(n)$, $O(d) = O(n^2)$, $O(c) = O(n^2)$, and $O(k) = O(n^2)$. So the worst case space complexity is $O(n^2)$. However, due to possible reductions for architectural space plans discussed in section 6.1.1, $O(d) = O(n)$, $O(c) = O(n)$, and $O(k) = O(n)$, so the expected case space complexity is actually $O(n)$.

6.3. Summary

The purpose of the task to compute dynamic constraints is simply to maintain the distance between two spaces that have come in contact with each other. That task dominates the computational complexity of the entire algorithm. But because it does

not affect the design object forces and has nothing to do with the basic function of the algorithm, the dominance of this step might be removed with an alternative implementation. One such alternative might be to merge nodes that have come in contact. For example, when two wall nodes are in contact with each other, one node is the 'master' and participates in the dynamic simulation while the other node is a 'slave' whose position is determined by the master. Since they more accurately work in concert and forces applied to the slave node are applied to the master, a better terminology might be 'representative' and 'citizens.' Additional contacts with either the representative or citizen node produces additional citizen nodes. Undoubtedly there exist other complicating factors, such as how to determine when two nodes are no longer in contact, or how to draw citizen nodes relative to representative nodes, but this alternative can potentially greatly reduce the computational requirements of the overall algorithm. Not only is the need to compute dynamic constraints eliminated, but the number of nodes is reduced as well, by as much as a half. In buildings with relatively long corridors, this can potentially cut the number of nodes in half.

7. DISCUSSION

An application is *useful* to the extent that it aids a user in accomplishing some task. An application is *useable* to the extent that its individual features work as expected. It is possible for an application to be useable but not useful for accomplishing a specified task. The goal is to make an application that is both useable and useful. This section discusses the approach as presented, to determine if it is potentially useful in practice.

I will begin with a discussion of the prototype implementation just described in section 5, “Prototype implementation,” and described in complete detail in appendix A, followed by some general observations about the concepts proposed in section 4, “A physically based approach to space planning.” These discussions only provide initial subjective observations based on the prototype implementation, which can be used to inform more rigorous future investigations.

7.1. Prototype

While using the prototype application during its development and implementation, a number of observations were made.

7.1.1. Objectives

The different types of design objectives needed to address most designer’s intentions are surprisingly few. One reason might be due to the limited number of geometric elements dealt with during development of the prototype. Another, more interesting, reason might be due to the definition of the problem as used here. The set of topological objectives is limited to the number of ways a vector can be applied to a point, that is, the number of ways a force can be applied to the center of a space.

Similarly, the set of geometric objectives is limited to the number of ways a force can be applied to the line node representing a polygonal edge. There are only so many ways a vector can relate to a point, which limits the number of possible design objectives. Although this may seem a shortcoming of my overall approach, within the domain of space layout planning the few objectives I have discussed enable the designer to produce a wide variety of results. It remains to be seen if these objectives are adequate as applied to the overall design process.

7.1.1.1. Alignment objective and importance of geometric objectives

Between topological and geometric design objectives, the geometric objectives are more critical from a design perspective. The physically based space planning approach is compelling to use during topological resolution, when a user can manipulate space relationships by dragging the space nodes around. But its potential power is revealed during geometric resolution when the user begins to apply geometric design objectives such as alignment objectives.

Because of this, the creation of the alignment objective during the development of the prototype was a defining moment in this research (see section 4.2.7.1 for its description and section 5.2 for more discussion regarding its value). With the ability to define alignment objectives, the user begins to truly *design*, by molding a particular plan into the design he or she has in mind or that is beginning to emerge from the explorations enabled by topographical manipulation.

Alignment objectives were intended to apply to parallel edges only. During their implementation, however, selecting non-parallel edges was not prohibited, so an inadvertent added feature is the ability to align one wall with the midpoint of another. With some simple modifications it could be made to align with any point. This provides an interesting demonstration of how potentially useful ‘features’ can

be serendipitously discovered during the software development process.

7.1.1.2. Unary topological objectives

Section 4.2.4 describes unary topological objectives, which are design objectives that apply a single force to a space node. The objectives described, including interior, exterior, and orientation, apply a varying force along a direction vector pointing either toward or away from the center of a set of spaces. The multiple passes of development and written description of these objectives provide an interesting example of the iterative nature of the process of research and problem solving.

The early prototype implemented these objectives by applying a constant force, which is counter to the requirements of a good design objective as described in section 4.2. The consequence of this implementation was that even when the objective was met there was still a force being applied to the space. This was a convenient way to implement and demonstrate the intent of these types of objectives, but it did not follow the general intent of a force-based objective, which is that once an objective is met, no forces are applied to its nodes.

The early written description of these objectives described this early implementation, and discussed their problems and limitations. In reviewing this description a potential solution came to mind, resulting in another round of implementation and the ultimate description presented in section 4, “A physically based approach to space planning.”

7.1.1.3. Relationship between topological and geometric objectives

As implemented, topological objectives have one method of behavior, they apply forces only during topological resolution, and are intended to be turned off during geometric resolution. The suggested process involves resolving first topological then

geometric objectives – once the switch is made to geometric resolution, topological objectives are turned off and geometric objectives are turned on. One problem with this approach is that it is possible for a topological objective to be achieved during topological resolution, but when the resolution method has switched to geometric resolution it is no longer achieved; namely, the circular representations are in contact with each other, but the polygonal representations are not.

One solution is to design topological objectives with two different modes, one active during topological resolution and another active during geometric resolution. For example, an adjacency objective could be defined in such a way that it applies no force when the edges of two spaces are in contact. In this way topological objectives can be active and measurable during geometric resolution.

7.1.1.4. Constraints and integral springs

Implementing design objectives as force applicators is a loose method of implementing design constraints, as mentioned in section 2.2.3.1. It is loose because the effect on an object at any one time is only the average of the set of forces applied to the object. A surer method of achieving design objectives, as discussed in a number of cases, is to use integral springs and dynamic constraints. An obvious question to ask is, why not use these methods for all design objectives?

One reason is the problem of dealing with over constrained systems. If all design objectives were implemented as integral springs, in an over constrained system a point in time will be reached when it is at equilibrium, but since some objectives haven't been met the force applied by the integral spring will continually increase. At some point the physical simulation will then be dealing with massively large forces and become highly unstable, and eventually the program will crash.

7.1.1.5. Strengths and level of importance

One potentially useful approach to using importances was discovered during development. Rather than displaying all objectives at the same time and requiring the designer to manipulate the plan with all of them displayed, which will usually require a high cognitive burden, one method is to use a hierarchical approach. First display only those objectives with the highest importance, and manipulate the plan until those are met, then iteratively display the successively less important objectives, with each iteration attempting to improve the plan. This process could even be automated in a manner similar to simulated annealing.

Much effort was expended to implement levels of importance, with the intent to allow the user to set different levels of importance for different design objectives. In order for this ability to be useful, however, changing the level of importance of a number of objectives by a significant amount should result in a significantly different result. This was not seen. Possible reasons are that 1) the averaging method inherent in the use of springs precludes this level of semantic fidelity, 2) the maximum strength of the springs was not high enough, or 3) the level of importance scale should be non-linear, such as logarithmic, instead of linear.

7.1.2. Modeless interaction

The prototype made extensive use of modeless interaction; the user could interact with most of the elements in the drawing area directly, without explicitly selecting a menu command. This kind of interaction was very useful and quick, and along with the active help could probably be learned by the user in a fairly short time. For small applications this methodology could be extremely useful. For larger applications such as a complete computer aided design application this methodology is probably not

as useful because of the large number of objects the user interacts with and the large number of commands used to interact with those objects. However, if a core set of objects and commands could be identified, and if that set could be ‘ordered’ into a logical and intuitive list, it is possible that this could increase the usability of even very complex applications.

7.1.3. Thumbnails

The thumbnails window provides a means to quickly save and restore a number of design states. This is a potentially very powerful tool. When working on designs during the conceptual phase, which is the main intent of the physically based space planning approach, it supports the ability to produce a larger number of design possibilities in a short amount of time.

7.1.4. Numerical integration

Runge-Kutta 4th order numerical integration was found to be more than adequate for the purposes of this prototype, and provides relatively fast computation with reasonable stability. In fact, the midpoint method proved to be quite stable in most situations. The Euler method, however, provided very unstable systems which invariably ‘blew up.’ The elements begin to oscillate so much that as a space plan they rapidly become meaningless and quickly leave the area of the drawing area.

Adaptive step-size methods provide a way to improve the accuracy of a dynamical simulation without greatly increasing the computational cost, and are essential for some physically based simulations where the accuracy of the behavior is important. In non-adaptive methods, the accuracy can be increased by reducing the time step between state computations (frames), at the expense of increasing the computation time for each frame. Adaptive step-size methods 1) compute a frame, 2)

estimate the error of the frame, 3) if the error is above a specified threshold the time step is reduced, 4) and continually loop through these steps until the error is below a specified threshold. I considered putting adaptive methods in the prototype for experimental purposes, but felt that they were not necessary for space planning in particular and probably other design domains in general. Physically based design seeks to make design elements *appear* to behave as dynamical elements, but because designers have no actual experience of perceiving them as dynamical they are not aware when they may not behave as accurately as they might.

7.1.5. *Worked examples*

During the vast majority of development, much of the focus was on getting the various parts of the system to work as originally envisioned, implementing the physically based simulation system and designing the individual objectives. The focus tended to be on the parts of the system rather than on the whole or how it would be used in a real design situation.

Although much manual manipulation was required to achieve the results shown in figure 5-7, it was obvious that time was saved by using the prototype. No detailed geometric manipulation was required, such as, “move this wall over here,” or “stretch this space over there.” The manipulation consisted of moving rooms around and the detailed positioning of the final result was taken care of by the physically based system. In other words, the designer’s thoughts were allowed to focus on macro level design actions rather than micro level command and geometric actions, thus supporting the supposition that this is a viable strategy for a design process.

However, there is still much work to be done before this approach can be used in a commercial application. Although the prototype works fast enough on a plan with half a dozen spaces, going to the residential plan used in the example in section 5.4

caused a noticeable slowdown in performance. Many necessary features or abilities are still lacking. These are described more fully in the concluding section as future work, but include multi-story support, hierarchical space structure, non-rectangular spaces, and circulation.

7.1.6. *Optimization*

The physically based space planning technique presented here *does not* provide an optimal solution to the space planning problem. An optimal solution is not possible given the nature of the technique; the final position of a room, for example, is a result of the average of forces being applied to it, which will rarely result in an optimal position. My position is that this is not a serious flaw in the technique, although some may argue otherwise. Its value is not in its automated nature, but in the *quality* of the interaction between the designer and the design that it adds to the design process. Instead of automatically producing an optimal solution, this approach looks for local optima and depends on the designer to recognize when a solution is weak and to make appropriate changes by hand to guide the system into a more optimal configuration.

It is in the nature of the space planning problem that it is NP-complete (Liggett and Mitchell, 1981b, p. 282), and the goal of many space planning techniques is to find optimal solutions. This is a natural goal to strive for given the apparent nature of the problem, its similarity to other layout problems, and the available tools used to solve them. However, not everyone believes that optimization is a valid goal. Lawson (1997) lends support to the position that optimization is not of prime importance by stating that “rarely can the designer simply optimize one requirement without suffering some losses elsewhere,” and that “there are thus no optimal solutions to design problems but rather a whole range of acceptable solutions . . . each likely to

prove more or less satisfactory in different ways and to different clients or users. Just as the making of design decisions remains a matter of judgment so does the appraisal and evaluation of solutions.” (Lawson, 1997, p. 123) Similarly, McCullough lends support to the position that quality is important by stating that “. . . the ultimate significance of postindustrial technology has to be in serving the need to work well – and not in automation,” and that “. . . it matters less what the technology can do alone than what you want to do with it.” (McCullough, 1996) Another argument is that an optimal solution implies the definition of an optimization function, and that function itself is not easy to define, and depends very much on who defines it. Each player in a design project, such as the client, user, or designer, would likely define a significantly different objective function, resulting in significantly different designs produced by an optimal automated design system.

This is not to say that physically based space planning cannot be enhanced or extended to provide a more optimal solution. For example, one potentially beneficial approach is to employ some form of simulated annealing to determine which set of initial space positions can produce an optimal topological arrangement, and then let the user refine the design using the physically based space planning interface. Another approach would be to integrate this technique with another optimal space planner that doesn't provide a manipulative interface. For example, a similar set of design objectives could be defined for the optimal space planner, which outputs an optimal plan. This potentially 'unimaginative' plan is then input into the physically based space planning program to allow the designer to manipulate it into a more aesthetically pleasing form.

7.1.7. *Essence of its usefulness*

What characteristics of the prototype were determined to be ‘essential?’ In other words, what features could be removed and still be left with a useful methodology?

The essence of this approach as used during topological resolution is as a ‘bubble diagrammer.’ One way to implement such an approach in a physically based system is simply to have all spaces wrapped with a virtual rubber band. While manipulating spaces the rubber band would keep them together into a single building, but still appear to ‘get out of the way’ when one space is moved through a number of others. None of the topological objectives would need to be modeled as force applicators, and their primary value would be in visually indicating to the designer whether or not objectives are being met.

One reason I believe the physically based space planning approach is as compelling as it is is because using and manipulating bubble diagrams is one of the first techniques learned by all architecture students. What architect hasn’t at one point in their career cut out pieces of paper for each space and moved them around in relation to each other?

However, this minimalist approach would remove the potential benefit of using each objective’s force to measure the ‘fitness’ of a particular design, to use Alexander’s terminology (Alexander, 1964).

Moving from topological to geometric resolution again reveals the power of using a physically based approach. It would be difficult to remove the force application implementation of geometric objectives such as alignment, area, and proportion. If that were done, these objectives would have no way of making the changes to a space plan that they are designed to make, and the process of design then begins to be similar to that of traditional CAD.

7.2. General observations

Here I discuss some general observations about the concepts proposed in section 4, “A physically based approach to space planning.”

7.2.1. *Graphic thinking, design exploration, drawing collaboration*

Three somewhat related concepts that characterize a useful design process are graphic thinking, collaboration, and design exploration. Graphic thinking involves how the elements of a design drawing or diagram are abstracted to have fuzzy semantic meanings to the designer, for example, “this line represents a wall in this rough position, instead of one face of a wall in a specific position.” Design exploration involves the process whereby the designer moves through the design space to visit an adequate number of possible designs. Drawing collaboration involves how the designer interacts with drawings and diagrams to move in a specific direction within the design space. One measure of the usefulness of the physically based space planning approach is to what degree it promotes these related concepts in the design process.

The benefits of using computer aids during the middle and later stages of design are well accepted. There is little dispute that they improve the process rather than hinder it. However, these statements cannot be made about design during the conceptual stage, as few computer aided applications truly support the quick, loose, sketchy freedom of idea generation required during conceptual design. Ware (2000, p. 381) notes the importance of supporting diagrammatic interaction:

Possibly the most challenging problem posed in data visualization systems is to support the way sketchy diagrams are used by scientists and engineers in the production stage. Discoveries and inventions that began as table-napkin sketches are legendary. Here is a description of the role

of a diagram by an architectural theorist (Alexander, 1964, p. 91)

Each constructive diagram is a tentative assumption about the nature of the context. Like a hypothesis, it relates an unclear set of forces to one another conceptually; like a hypothesis, it is usually improved by clarity and economy of notation. Like a hypothesis, it cannot be obtained by deductive methods, but only by abstraction and invention. Like a hypothesis, it is rejected when a discrepancy turns up and shows that it fails to account for some new force in the context.

It is clear that if creativity is to be supported, the medium must *afford tentative interactions*. Imprecise, “loose” sketches gain from a lack of precision that affords multiple interpretations. The fact that a line can be interpreted in different ways . . . can be a distinct benefit in enabling a diagram to support multiple tentative hypotheses. The sketches people construct as part of the creative process are rapid, not refined, and readily discarded. (emphasis added)

The automated nature of a physically based space planning approach affords tentative interactions because it removes much of the precise interaction required of the user, who thinks at the level of “I want to move this space somewhere over here,” rather than “I want to move this space to this specific position.” This characteristic strongly suggests that the approach enhances design exploration, because the designer can produce a large number of designs in a small amount of time.

The physically based nature of the approach active during direct user interaction strongly suggests that it enhances design collaboration, because of the ‘tactile’ quality of interacting with its elements in an experientially familiar physical manner.

There is less support, however, to suggest that the approach enhances graphic thinking. Walls remain semantically walls, and spaces remain spaces during manipulation and design analysis, and never ‘jump’ to other meanings during the design process, as happens with hand-drawn diagrams. Possible exceptions might be during topological resolution when spaces overlap and if design objective forces are displayed as shown in figure A-21. These non-reality based elements displayed in the plan may invoke semantically different meanings in the mind of the designer and may spark new and interesting design possibilities.

7.2.2. Automated space planning vs. a manual interactive experience

The inspiration for and much of the work toward the proposed approach was in its automated potential. The original intent was to provide a means to automate the space planning process. However, compared to the many other methods of automated space planning, this method suffers from many problems, and indeed cannot automatically produce very good space plans.

However, once the prototype was essentially working, the benefits of this approach shifted from its automated nature to the manipulative experience it can give the user. It *does* automate parts of the process, but it is valuable not because of that, but because the experience of working with the space plan in a physical and manipulative way is so compelling. The user is encouraged to play with the design, explore new ideas, discover new relationships.

7.2.3. Newtonian physics vs. modern physics

A natural question to ask given the hypotheses presented here is “If Newtonian physics can be applied to design imagery, why not more advanced physics theories such as Einstein’s theories of relativity or quantum mechanics?” After all, these more

modern theories have experimentally proven to be more representative of reality than has Newtonian dynamics.

A reply to this question is that modern theories may be more representative of reality, but not at the everyday level of reality that humans can perceive and intuitively understand. This assertion is supported by none other than Stephen Hawking:

Einstein's general theory of relativity predicted a slightly different motion from Newton's theory. The fact that Einstein's predictions match what was seen, while Newton's did not, was one of the crucial confirmations of the new theory. However, we still use Newton's theory for all practical purposes because the difference between its predictions and those of general relativity is very small in the situations that we normally deal with. (Newton's theory also has the great advantage that it is much simpler to work with than Einstein's!) (Hawking, 1988, p. 10)

Newtonian dynamics attempts to explain the motion of bodies as a result of forces acting upon them. It does not, however, fully explain the motion of bodies. Einstein's General Theory of Relativity encompasses Newtonian dynamics in explaining more of the observable phenomena of motion. Despite the fact that Newtonian dynamics does not *fully* explain the motion of bodies, it is adequate in explaining that motion operative at the experiential level of humans. We cannot observe with our own senses light bending around stars due to the curvature of space, so this fact has little, if any, impact on the way we interact with our environment. Our senses, however, *are* able to observe the motion of bodies due to gravity, how they react to collisions, how they are affected by the viscosity of air and water, and many other phenomena that Newton described. These 'facts' form the basis for our experience

of and understanding of the physical elements of our environment.

7.2.4. Possible reasons for being compelling

When presented informally to architects, educators, students, and even lay people the prototype application is almost universally well received. It is conceptually very easy to understand the intent of the approach from the simple statement “imagine moving spaces connected with springs,” and very easy to envision how this approach might be used to aid in solving the space planning problem. What might be some reasons for the positive reaction?

One reason may be that it is possible people are drawn to a physically based approach to space planning because they are unfamiliar with existing approaches, which may be just as exciting to them as this one. Despite four decades of research into automated space planning, few commercially available software packages employ the results of this research, and no widely used software packages use automated space planning techniques. It is likely that the majority of architectural educators and virtually all architectural students are unaware of any approach to automated space planning. At conferences where the majority of attendees are researchers in this field, most of them are probably aware of many approaches to automated space planning. For these people though, their cognitive exposure to these ideas is probably limited to reading technical papers and viewing author’s presentation, and not with having first hand experience with using prototype software applications. In this sense, when confronted with an application that appears to work, they may still be considered to *not* be familiar, at a design level, with other approaches to space planning.

Another reason may be that people are drawn to the ‘WOW’ factor, that is, to the novelty of the software. Few people, even CAD researchers, have been exposed

to physically based modeling, and fewer still understand the principles behind its use. And none have ever seen the rooms and walls of a space plan moving around dynamically, as if they were alive. In their experience, elements move because they, the designer, are directly causing them to move through some elementary graphic command in a CAD program. The novelty of *viewing* a dynamical space plan may not transfer to *using* a physically based space planning tool to solve real design problems.

A third reason may be that manipulating design objects as if they are separate physical entities with their own mass and substance is directly related to our experience of manipulating real world objects. We are intimately familiar with how real objects move and relate to others under the influence of environmental forces, so it may not be such a stretch to see and interact with design objects that behave similarly. This reason does support the use of a physically based approach to design.

A fourth reason may be that designers are interested in using this approach because it is closely related to how they *think* about designs. If it can be shown that a designer's mental imagery during the process of design behaves as a dynamical system, then a computer aided design tool that presents design elements as dynamical elements should be of compelling interest to designers. A first outline of this theory is presented in section 8, "Theoretical Implications."

The last two reasons, if accurate, provide support for a physically based approach to design, while the first two reasons do not.

7.2.5. *Holistic nature*

One of the significant characteristics of this physically based approach to space planning that became apparent after reviewing a number of other approaches is its *holistic* nature. The hypothesis of this dissertation is that the space planning process can be

modeled using physically based mechanical metaphors. In this work I have shown that space plan elements and design objectives can be defined as virtual mechanical objects, the collection of those elements can be used in a dynamical system to compute a space planning solution, and then the user can interact with the solution to modify it by applying forces that are themselves models of the same mechanical objects. *Representation, computation, and interaction are all encompassed within the same dynamical system.*

It is highly doubtful that any other approach to automated space planning can make this claim. The physical elements of the plan are usually represented in some geometrical or spatial manner, and the design objectives are usually represented in a completely unrelated manner. The computation of a potential space plan given these representations is usually highly complex and highly iterative in nature. Finally, in the rare event that a particular method accommodates user interaction, it is usually implemented using basic geometric CAD commands such as move, stretch, scale, etc.

For example, iterative approaches such as Weinzapfel and Handel (1975), Liggett and Mitchell (1981b), and Akin et al. (1988), where a design solution is changed from one state to another by iteratively evaluating one or more spaces, are not holistic. Each space is considered only in relation to the spaces already placed, and the order in which the spaces are evaluated affects the final solution. In contrast, the approach proposed here considers *all* spaces and *all* design objectives simultaneously while computing the next design state.

In physics and mathematics simple theories are thought to be beautiful and often prove to be correct relative to more complex theories. As Albert Einstein says “things should be made as simple as possible, but not any simpler.” A related sentiment in the field of design, by Antoine de Saint-Exupery is “you know you’ve achieved perfection in design, not when you have nothing more to add, but when you

have nothing more to take away.” So, even if it turns out that the approach does not have the practical application that we hope it does, its holistic nature seems to be an important characteristic in itself in illuminating the design process.

7.3. Summary

I feel that my prototype system provides a convincing demonstration of the attractiveness of the responsive design approach and the use of physically based methods in implementing this approach. The prototype effectively demonstrated the concepts defined in section 4, “A physically based approach to space planning,” and revealed the importance of a highly manipulative user interface for the designer and the benefits of using a physically based approach to provide that manipulation. All of these observations remain subjective in nature, however, and will require much future empirical research to either support or refute them.

The work presented here may have theoretical implications beyond simply a new model for space planning, and may reveal something new about design cognition. In the next section I will discuss an interesting though speculative theory of ‘dynamical design imagery,’ which proposes that, rather than being static ‘pictures,’ the elements in a designer’s mental imagery during the act of design are dynamic in nature and act as a dynamical system.

8. THEORETICAL IMPLICATIONS

While speculating on why the approach presented here was compelling to many viewers, a theory emerged that may describe cognitive processes active in a designer's mind during the act of design. This theory, which I call *dynamical design imagery*, postulates that the mental images in a designer's mind during the act of design are dynamical in nature and that the elements of those images behave as elements in a dynamical system.

If this theory can be shown to have validity, it has significant implications for the development of computer-aided design applications. A designer's desire to use one tool over another might suggest that there is a correlation between the way the tool works and the way the mind works, that is, there is a better fit between tool and mind. So, if it can be shown that an element of physics exists in cognitive design processes, tools that respect that element, use it, and support it, are more likely to be easier to use, and by extension produce better results.

8.1. Dynamical design imagery

For millennia our view of the intermediate products of the design process, drawings, has been essentially static. This view is partly due to the fact that the final products of design, such as a building in the area of architectural design, tend to be static. It is also due to the technological tools of the time, whatever that time may be before the present. Drawings are themselves static, whether produced with pencil and paper or more recent computer aided design tools. Even when viewing unconstructed buildings with animation tools such as virtual walk-throughs, the elements of buildings are still cognitively understood as being static.

Dynamic terminology is often used to describe the images in designer's minds

during the process of design (such as Arnheim (1977) and McKim (1972)). In these descriptions this dynamic terminology is limited to notions of movement, without an underlying understanding or description of *how* the movement is accomplished. Yes, design imagery is dynamic. But how is it dynamic? What does it mean that the images are dynamic? What additional insight can this idea bring to the design process? How can design tools be created to support a dynamic view of design imagery? Can such tools improve the quality of the design process and of designs?

Is there more to dynamic design imagery than simple movement or change?

A theory began to emerge during the development of this dissertation that begins to include notions of dynamic systems in the cognitive design process. I call this theory *dynamical design imagery*. *Dynamical*, because it contends that the elements of the cognitive *design* process behave as a dynamical system; not simply 'dynamic,' which connotes 'change' instead of specifically 'change based on physical dynamics;' and *imagery*, because the theory applies to the mental representations of the focus of the design problem, not on the physical representations, such as drawings or models. Some hypotheses that result from this theory are that design imagery is dynamical in the sense that its elements behave as a dynamical system, that viewing design imagery as a dynamical system can improve our understanding of the design process, and that design tools that treat design elements as dynamical provide a better fit between the manipulable and mental elements of design.

Some have hinted at the dynamical character of mental processes. McCullough references Focillon (1934) in his discussion of *The Phenomenon of Handicraft*;

Focillon addresses the *dynamics of creativity*, for which he argues that art must be tangible. Object form, he asserts, is the one way to record the *flux of forms* that occurs in space, in matter, and *especially in the*

mind. The apprehension and giving of form is a dynamic process, rather than a static code; giving form gives works their meaning. Of course the givers of form are the hands. (emphasis added) (McCullough, 1996)

Bohm and Peat expresses how imagination can be thought of in physical, dynamic terms:

Literally imagination means ‘the ability to make mental images,’ which imitate the forms of real things. However, the powers of imagination actually go far beyond this, to include the creative inception of new forms, hitherto unknown. These are experienced not only as visual images, but also through all sorts of feelings, tactile sensations, and kinesthetic sensations and in other ways that defy description. The ability of Mozart and Bach to sense whole musical works all at once could be regarded as a kind of musical imagination. The activity of the imagination does not therefore resemble a static-picture but rather it is closer to a kind of “play” that includes a subtle orchestration of feelings, as well as a sense of intention and will. (Bohm and Peat, 1987, p. 261–262)

There is empirical evidence that some mental processes are analogous to physical processes. If a person has stored a mental image of a familiar object viewed from a set orientation, and is viewing a similar object at a different orientation and trying to determine if they are the same, she will mentally rotate the viewed object until it aligns with the stored image. The interesting thing is that the greater the difference in orientation of the viewed object to the stored object, the longer it takes to perform the mental rotation. In this case, the mental process of rotation is similar in function, application, and time to the physical process of rotation. Figure 8-1 shows some of the figures used in an experiment. During the first part of the experiment subjects

were taught a number of figures in set orientations, such as that on the left. Then they were shown the same figure but at a number of randomly selected orientations. It was found that the “the farther it would have to be rotated to be aligned with the nearest familiar view, the more time people took.” (Pinker, 1997, p. 280)

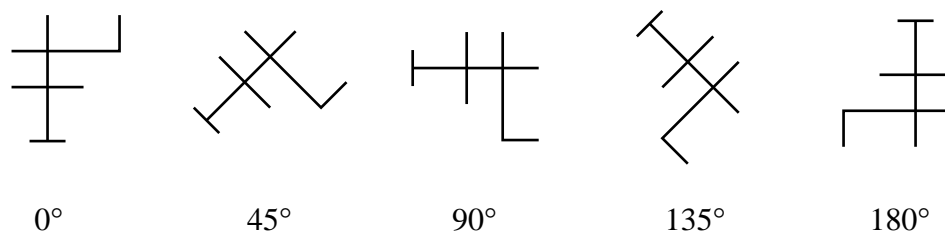


Fig. 8-1. Mental rotation [redrawn from (Pinker, 1997, p. 281)]

I present a number of arguments to help support a theory of dynamical design imagery. One draws parallels between the physical process of modeling clay and the mental process of design. Another provides a direct argument using recent theories of cognition. A third, more indirect, argument attempts to incorporate existing models of the natures of design, cognition, and dynamics into a framework that sequentially leads to a dynamical model of design imagery from a number of different directions.

8.2. Clay argument

The simplest and probably most compelling argument for a theory of dynamical design imagery comes from working with clay. Just as a mass of clay is changed from one state to another with the application of physical forces, designs in the designer’s mind may change from one state to another with the application of ‘design’ forces and mentally follow a similar ‘physics’ process.

In terms of movement through design space, a parallel can be drawn between a clay model in physical space and an architectural model in mental space or mental imagery.

In modeling clay, changing its shape by applying forces with hands and tools follows a continuous path through physical space from one shape state to another. Even adding and subtracting chunks of clay are continuous processes. The design state of the clay can only change with the application of physical forces.

Do these same rules and processes apply in some way to design imagery? A designer may have one mental image of a design in a particular state which probably has a number of features that do not meet the goal state, so the designer's objective is for the one state to change into another. It might be possible that a dynamic mental process takes place whereby one state is transformed into another while maintaining continuity in the mental design space.

8.3. Cognitive science argument

A direct argument that design imagery is part of a dynamic system comes from recent research in cognitive science based on the *dynamical hypothesis*, which says that "cognitive agents are dynamical systems" (van Gelder, 1998). If the cognitive processes of the brain can be shown to be part of a dynamical system, then a specific cognitive process, such as design, is also likely to be a part of a dynamical system. Then an effort should be made to explain particular aspects of the design process in terms of a dynamical system, as well as explain how design tools can be created to support this mental model of design.

The prevailing explanation for how human cognitive processes work is called the computational hypothesis. The computational hypothesis says that "cognitive agents

are digital computers” (van Gelder, 1998), which is based on the work of Newell and Simon (1976). It says that cognitive agents, such as humans or computers, are *physical symbol systems*. They store quantitative bits of information about their environment, and use an algorithmic process to modify those stored values, resulting in a change in the values and some type of action in the environment.

In contrast, the dynamical hypothesis says, as stated before, that cognitive agents are dynamical systems. The difference between these hypotheses is best explained by contrasting the two.

The computational hypothesis recognizes that states change *over* time, but time itself is not a quantifiable variable. In the dynamical hypothesis, time is a quantifiable, essential element of a dynamic system, in that its state changes *in* time. Thus, the rate of change of state becomes an essential characteristic.

In the context of a state space, which is the hyper-space of all possible states of a ‘system,’ the computational hypothesis recognizes the *difference* between states. However, the dynamical hypothesis recognizes the *distance* between states.

The dynamical hypothesis recognizes that cognitive processes are *situated* or *embedded* in a context; that is, the environment plays an integral role in cognitive processes and can’t be excluded. The computational hypothesis does not account for environmental context.

The computational hypothesis says that representations are a key factor in the cognitive process. The dynamical hypothesis treats representations as another element in a dynamic system, and says that representations can change in time as any other quantifiable variable, and in fact may not be present at all.

A closer look at these differences between the computational hypothesis and the dynamical hypothesis reveals that the dynamical hypothesis contains all elements of the computational hypothesis, but adds additional considerations. The relation-

ship between these two views can be compared to that of the physics of Einstein and Newton; Einstein's theory of relativity does not replace Newtonian dynamics, but includes it and explains even more natural phenomena than does Newton. For example, the dynamical hypothesis extends the computational hypothesis notion of time to include rates of change in time; the dynamical hypothesis extends the computational hypothesis notion of the difference between states to include the distance between states; the dynamical hypothesis accounts for the environment where the cognition occurs whereas the computational hypothesis does not; and finally, the dynamical hypothesis extends the computational hypothesis notion of a representation to include the possibility of representations themselves changing over time.

The dynamical hypothesis says something about the space around a state. The computational hypothesis describes a state and how it might be different from other states, but with the dynamical hypothesis, seeing a state in motion says something about the 'landscape' around the state, revealing more information.

Because the dynamical hypothesis encompasses all of the phenomena of the computational hypothesis and more, it should, in theory, explain more cognitive phenomena and be able to be applied to more cognitive situations.

If cognitive processes can be shown to be dynamical systems, then it probably holds that specific cognitive processes are also dynamical systems (as long as the specific process is not at too elementary a level). The act of design by a human is a specific cognitive process. Therefore, based on the dynamical hypothesis of cognitive science, the act of design by a human is a dynamical process. The act of design involves a wide variety of information types, such as textual, graphical, quantitative, and so forth. By extension of the previous argument, the act of processing those information types is also a dynamical process (again, without decomposing into atomic elements). The goal of the design process, at least for the definition of

design as discussed here, is a building or other geometrically defined entity. The focus of the design process during the act of design is a representation of that entity that uses graphic elements. Therefore, the graphical representation of the goal of a design process is a dynamical system.

Van Gelder supports the application of the dynamical hypothesis to sub-processes of cognition when he says,

In the prototypical case, the dynamicist focuses on *some particular aspect of cognition* and proposes an abstract dynamical system as a model of the processes involved. The behavior of the model is investigated using dynamical systems theory, often aided by simulation on digital computers. A close match between the behavior of the model and empirical data on the target phenomenon confirms the hypothesis that the target is itself dynamical in nature, and that it can be understood in the same dynamical terms. (emphasis added)

‘Design’ is the “particular aspect of cognition” on which the dynamical hypothesis will be focused.

A greater understanding of the design process as a dynamic system, and the application of that understanding to the design of tools that aid in the design process should yield better design tools.

8.4. Design-mind-motion argument

The Design-Mind-Motion argument shows that successive combinations of three fairly distinct major concepts lead to dynamical design imagery.

Figure 8-2 shows overlapping circles, each circle representing three major concepts of Design, Mind, and Motion. Overlapping areas of two circles yield combi-

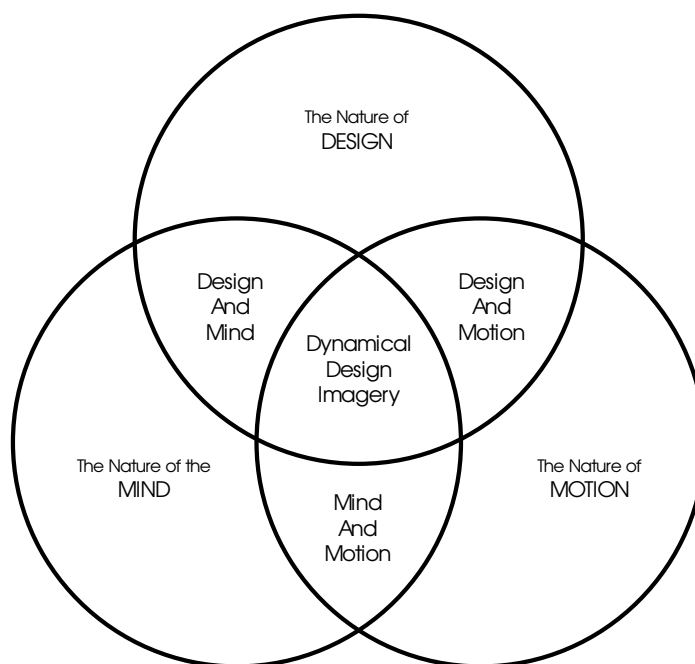


Fig. 8-2. Dynamical Design Imagery - the three major concepts of Design, Mind, and Motion

nations of the two concepts, and overlapping areas of all three in the center yield Dynamical Design Imagery. The argument 'flows' from the outside to the inside, showing that combinations of a large number of existing theories and ideas can lead to a theory of Dynamical Design Imagery.

It may appear that the choice of the three major concepts is arbitrary, but it can be argued that they represent aspects of broader, unique, fundamental concepts. There are many ways of describing these concepts in terms that are coherent to each other. Separating concepts into three major areas seems to be a common occurrence, and table 8-1 lists some. It is interesting to relate each triad relative to each other, as was intended when each was originally defined, but it is also interesting to look at each column and see the similarity between them.

The Mind is the most irreducible concept of the three. It represents cognition,

Table 8-1. Similar Mind/Motion/Design separations

Mind	Motion	Design	
Mentality	Physicality	Intentionality	
Mind	Body	Spirit	
Awareness	Existence	Action	
Consciousness	Reality	Causation	
Consciousness	Reality	Thought	(Combs, 2003)
Intellectual	Physical	Emotional	Biorythms
Significance	Soma	Energy	(Bohm, 1995)
Meaning	Matter	Energy	
Subtle	Manifest		
Viable	Capable	Desirable	(Cooper, 1999, p. 72)
Business	Engineering	Design	
Exotic	Dorsai	Friendly	(Dickson, 1959)
Mind	Brain	Chaos	(Pinker, 1997, p. x)
Perception, Reasoning	Social Relations	Emotion	

thought processes, awareness of existence, logic, etc.

Motion is the most reduced concept of the three. It actually is just one aspect of existence, or physicality. Motion is an essential characteristic of physical objects. It can be argued that the existence of objects is unimportant if some notion of their interaction is not taken into account - objects exist not in themselves, but in their interaction with others.

Design represents the process whereby objects that exist are rearranged or modified with some purpose in mind (or Mind, in this context). A word that aptly describes the concept of Design is *intentionality*. Design is the connection between the Mind and the Body.

At a higher, more philosophical level, it can be argued that these three major concepts are inseparable, and that the three circles representing each fully overlap. Mind cannot exist outside a physical context and has no purpose if it has nothing to Design. Design cannot occur without Mind to guide it and without Motion on which to operate. However, this argument breaks down with Motion. Motion and

physicality *can* exist without Mind or Design, which may indicate that the triad of concepts is not adequately formed.

It is useful nonetheless to describe each concept as a separate entity, thereby enabling understanding and insight of new concepts from different directions and perspectives. Separating and understanding the major ideas, and subsequently combining them back together again, leads toward the argument for Dynamical Design Imagery. I recognize that choosing which concepts are relevant in the combinations represents a type of ‘spin’ placed on the argument, and that other choices may yield different, possibly conflicting, though valid results.

With this overlap of major concepts it is inevitable that some ideas discussed relative to one may also be applicable to others. For example, decision making is an integral part of the Mind and how an entity interacts with its environment. But it is also an integral part of the nature of Design; design is a process of solving problems, which necessarily includes an element of decision making.

Figure 8-3 shows the same diagram as that shown in figure 8-2, with some characteristics of the three major concepts listed. These characteristics are not complete, and may not even be the most important ones for their respective concepts, but are meant to be a starting point for future comparisons.

Characteristics of the nature of design include design states, a single instance of a possible design; design space, the space of all possible designs for a given design problem; and design processes, the wide variety of methods designers use to arrive at design states within a design space.

Characteristics of the nature of the mind include cognition, imagination, and learning.

Characteristics of the nature of motion include physics, our mathematical and conceptual understanding of how bodies move in relation to each other; simulation,

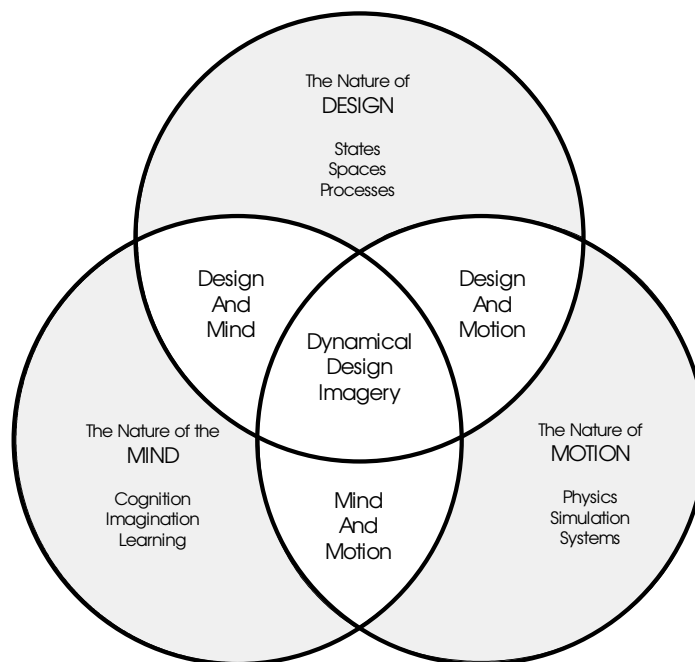


Fig. 8-3. Dynamical Design Imagery - some characteristics of the three major concepts

ways to more simply represent the complex motion of bodies; and systems, ways to define and work with a set of bodies.

Given the three major concepts just described, we can show that a wide variety of theories and ideas are combinations of two of these concepts. Figure 8-4 shows the same diagram as that shown in figure 8-3, with some theories and ideas listed in the three different overlaps of each combination of the three major concepts. These theories and ideas will only be mentioned here to give an idea on how the Design-Mind-Motion is structured. A more thorough description is left for future work.

Combining characteristics of the mind and of design yields theories such as perception, visual thinking, graphic thinking, reflective thinking, and spatial thinking. Combining characteristics of design of of motion yields theories such as dynamic form, dynamic behavior, design simulation, and design exploration. Finally, combining characteristics of the mind and of motion yields theories such as kinesthetic

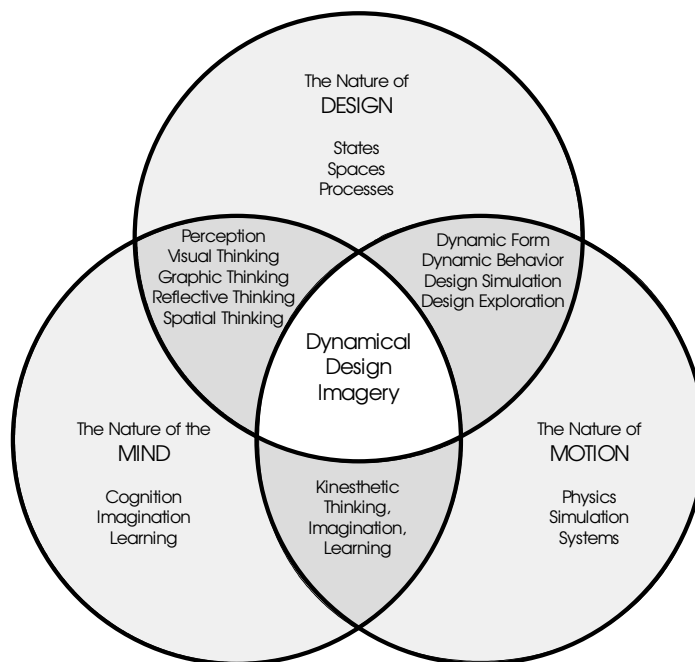


Fig. 8-4. Dynamical Design Imagery - theories as combinations of two major concepts

thinking, kinesthetic imagination, and kinesthetic learning.

Figure 8-5 shows that the theory of Dynamical Design Imagery is a combination of all three major concepts, as shown in the overlap of all three circles. As the three arrows in the diagram attempt to show, when the theories and ideas of the dual combinations are in turn combined with the missing major concept, dynamical design imagery emerges.

Graphic thinking, a combination of mind and design, is a theory proposed by Laseau (1980) that the sketch itself and the designer's mind work in concert with each other, and that the act of sketching is an important part of the process. Add the missing concept of motion to this idea, and you have sketches that are in motion, and the designer thinking about and interacting with the elements of those sketches using tools that support that motion.

Design exploration can be thought of as a combination of design and motion,

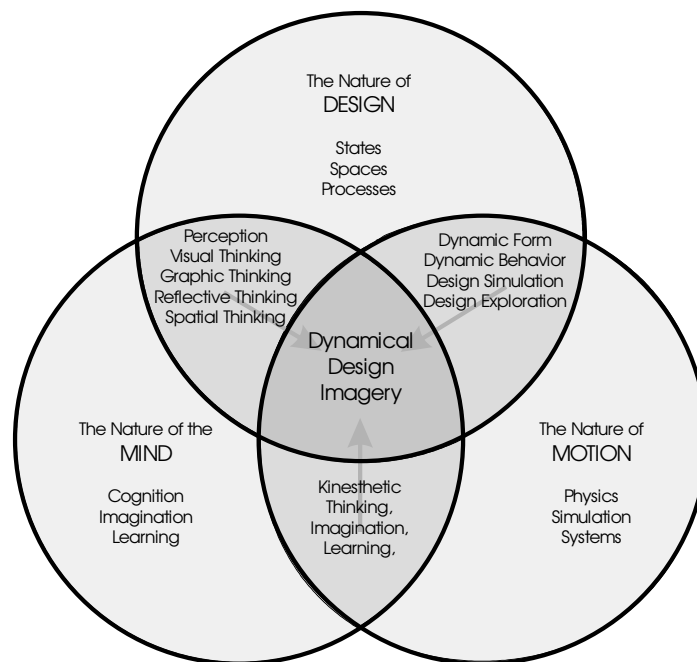


Fig. 8-5. Dynamical Design Imagery - a combination of all three major concepts

in that it is movement through a design space. Add the missing concept of the mind to this idea, and you have thinking about design as movement, the cognitive process of design exploration with an intuitive understanding of how a design space is structured and how to move from one state to another within that space.

Kinesthetic imagination, a combination of the mind and motion, is the ability to imagine the motion of objects. When we think or dream about objects they typically move in our minds while obeying the laws of physical motion. Add the missing concept of design to this idea, and you have kinesthetic imagination applied to the design process, imagining the elements in the design process as moving in relation to each other, and obeying their own physical laws.

These are just three of the many possible paths from a single concept of mind, motion, or design, to a combination of all three, all of which can be thought of as

some aspect of dynamical design imagery.

8.5. Summary

During the development of the ideas presented in this dissertation, a theory emerged that the mental constructs of a designer's mind during the process of design are part of a dynamical system. Three arguments to suggest support for the theory were presented. The first argument is fairly direct and intuitive, and compares the cognitive design process to the act of modeling clay - "the design process 'is' like modeling clay." The second argument is also fairly direct, although much less intuitive, and is based on recent theories in cognitive science - "a cognitive process 'is' dynamical in nature rather than computational in nature." The third argument is much more indirect, and attempts to encompass many existing theories by showing that they are combinations of the three major concepts of mind, motion, and design, and that dynamical design imagery is a combination of all three.

This theory is admittedly highly speculative, but nonetheless interesting. As stated earlier, if it can be shown to have some validity it will have significant implications for the design of computer-aided design systems. An enormous challenge remains in designing appropriate experiments to produce empirical evidence that the theory is correct.

9. CONCLUSIONS

The primary assumption of this research is that a digital design tool based on a physics paradigm can facilitate the architectural space planning process. This assumption lead to these hypotheses: Newtonian dynamics can be used

1. to define mechanical metaphors to represent the elements in an architectural space plan,
2. to compute architectural space planning solutions, and
3. to interact with architectural space plans.

To address these hypotheses I have described and implemented a physically based space planning methodology, showing that space plan elements and design objectives can be defined using mechanical metaphors, the collection of those elements can be used in a dynamical system to compute a space planning solution, and the user can interact with the solution to modify it by applying forces that are themselves models of the same mechanical objects. I found that an important characteristic of this approach is that representation, computation, and interaction are all defined using the same paradigm. This contrasts with most approaches to automated space planning, where these three characteristics are usually defined in completely different ways.

These hypotheses raised two questions. The first question was one of representation: How can the elements in a space planning design problem be modeled with mechanical analogues? Given this representation, the second question was one of implementation: How does an implementation of this representational model work?

In answer to the question of representation, I found that it was possible to model the elements of a space plan using physically based techniques, despite the fact that

there was not a one to one correspondence between what the technique was meant to simulate in the physical world and how the design element behaved in the mental world. For example, a wall is not a point mass, yet a point mass was used to represent a wall in the design simulation. Some of the design objectives, such as exterior and interior objectives, took a number of iterations before they were reasonable. But for the most part there was an appropriate physical analog for design elements and objectives.

In answer to the question of implementation, I successfully developed a prototype software application that implemented the design elements as they were defined.

Based on a subjective evaluation of the prototype implementation, I found that it demonstrates a feasible process for producing space plans, and that it has the potential for improving the design process due to quality of the manipulation that it provides to the designer, and the potential for greatly improving design exploration. As implemented the proposed approach was somewhat computationally expensive and may not scale to work effectively on large design problems. However, the results of the computational complexity analysis revealed some ideas that should greatly increase its performance. Much further study is required to determine if these results apply as described to actual design problems.

9.1. Contributions

The primary contribution of this work is a mapping from architectural space planning concepts to physically based metaphors of mechanical objects. The proof of the success of this mapping is a working software application, and a demonstration of its use in creating a sample architectural floor plan.

This work makes a number of unique contributions:

- It defines physically based representations for a number of architectural design objectives, using mechanical objects such as springs and force fields as metaphors.
- It defines a methodology for using these design objectives in a physically based simulation to compute solutions for architectural space plans.
- Along with these contributions of physically based representation and computation, it uses physically based direct manipulation to provide a means for a designer to interact with a space plan by modifying existing objectives and adding new ones. These three elements, representation, computation, and interaction, are holistically encompassed within a single paradigm.
- It describes the implementation of these elements in a prototype software application that demonstrates the potential usefulness of the approach.
- It proposes that the familiar physical nature of design elements during the process of design helps the design process – by being interactive, compelling, interesting, responsive, and explorative.
- It introduces a potentially important theory of *dynamical design imagery*, which may provide insight into some characteristics active in the brain during the process of design.

9.2. Future work

The proposed physically based approach to modeling design objectives is a previously unexplored concept, and raises many new questions and presents many opportunities for future work. Some of these are obvious and require answers and elaboration in

order to make this approach truly useful to space planners. Other questions are of a more theoretical, fundamental nature.

9.2.1. Extensions

The concepts presented in section 4, “A physically based approach to space planning,” and their implementation in the prototype are only a start. There exists much opportunity to extend these concepts to encompass greater functionality and usability. Some extensions, such as multi-level and circulation problems, are natural ones and often the first questions asked by architects after observing a demonstration. Other extensions, such as using forces to ‘measure’ the fitness of a plan, take advantage of the physically based nature of the approach and represent new and interesting opportunities.

9.2.1.1. Other design objectives

The most important design objectives and the ones most required for space planning problems were described in the previous sections and implemented in the prototype. Here are some examples of other possible design objectives, and there are no doubt more.

A gravity objective would be a type of geometric objective that attempts to remove gaps between adjacent spaces by making them ‘gravitate’ toward each other. Such an objective might solve the problem of removing undesired holes in a plan.

Field objectives could be defined as a grid of values placed on the plan site, and could apply forces to nodes depending on their location within the field. They are similar to what Grant (1983) calls ‘nature-of-the-spot.’ As Grant describes, they could be used to model physical conditions such as soil type, economic conditions such as land cost, or social conditions such as quality of a school district.

Another possible design objective is what Grant (1983) calls a ‘gestalt’ objective, such as formality/informality. For example, these could be used to impose symmetry or balance to a space plan.

All the design objectives described in section 4 and the potential new ones described above are continuous in nature. However, there exist other kinds of design objectives that cannot be adequately defined continuously and are discrete. Harada (1997) has proposed one solution for handling the physically based manipulation of discrete changes, but it might be difficult to represent such discrete design objectives in my implementation. Although it might be fairly easy to define a discrete objective itself as a force applicator, the interaction of such objectives with others might lead to unsatisfactory behavior. For example, a discrete objective might have two valid states, *A* or *B*. If at one point in time it is in state *A*, other forces might cause it to switch to state *B*, but then the same forces might cause it to switch back to state *A*, thereby causing part of the plan to oscillate.

9.2.1.2. Multi level problems

The prototype application only works in two dimensions, and one of the first questions asked by anyone presented with it is “how will this work in three dimensions?” Physically based modeling systems typically work in three dimensions, so making the system work from a mathematical perspective is relatively trivial. The issue, however, is the nature of the space planning problem, which is inherently planar. Except for unusual cases such as ramped parking facilities and banked velodromes, most floors in a building are level. This design limitation can be used to advantage when applying the concepts of physically based space planning to multi-story buildings.

The multi-story space planning problem is essentially a series of two dimensional problems. Each floor or sub-floor in a building can then be viewed as a separate,

two dimensional physically based space planning system.

Multi-floor spaces can be defined as a single space that can appear in multiple systems. Prime examples are elevator and stair shafts. The elevator space can appear in all floors of a building design, and any design objectives applied to the elevator may cause the elevator to change location on *all* floors.

Vertical forces can be applied that influence on what floor the space is located. For example, if most of the rooms on one floor are under their desired square footage, while most of the rooms on the floor above are over their desired square footage, vertical forces may knock rooms from the upper floor to the lower.

9.2.1.3. Circulation

Corridor spaces can be simulated by drawing long thin spaces, but their circular representation isn't accurate during topological resolution, which then requires much manipulation during geometric resolution. A possible solution is to represent corridor spaces as a series of line nodes rather than a single point node as described in section 4.1.3, and its area could be defined with a width, instead of both a width and depth.

9.2.1.4. Hierarchical space structure

As described in sections 4.1.3 and 5.1.3, the space object was designed to represent a generic space in a hierarchical structure. The prototype space data structure accounted for the hierarchy, but was not implemented beyond a single hierarchical level. Interesting issues would inevitably surface if they were to be implemented, especially regarding how the designer interacts with spaces at different levels of the hierarchy.

Another issue is the definition and use of a unique shape surrounding a space in

one level of the hierarchy that contains spaces within it, as shown in figure 4-10.

9.2.1.5. Non-rectangular spaces

The use of rectangular spaces is a useful limitation when developing space planning methodologies, as mentioned in section 5.1.1. Allowing a variety of space shapes other than rectangular would be necessary in any useful space planning tool. As described in section 4.1.2, defining shapes in a physically based space planning systems is not exactly straightforward, and a number of issues peculiar to this approach are bound to arise.

9.2.1.6. Functional strain

When parts of a physically based space plan are over-constrained and the plan is at equilibrium, design objectives are still applying forces to nodes attempting to move them. The magnitude of these forces is information about the plan that can potentially be put to use as a measure of the ‘fitness’ of a plan.

The **potential energy** of a body is its energy of position, and might be used to measure a design’s functional strain; that is, to measure how far off a design is from its set of design objectives. If a plan completely satisfies all design objectives it will have a functional strain of zero. Otherwise it will have a positive functional strain. Two plans can then be compared by evaluating this measure – the plan with the lower functional strain value is a ‘better’ design than the other one.

The potential energy of a body is a function of all the forces acting on it, and it is possible that a body has a potential energy of zero even when a number of large forces might be acting on it. For this reason potential energy itself cannot be used as a measure of functional strain. One solution to this problem is to take the sum of the potential energy, and better yet the square, of each individual design objective.

This is a classic measure used in optimization and has the effect of de-valuing small errors and over-valuing large ones.

Given a methodology for measuring the fitness of a plan, two plans with a similar set of design objectives can be compared. This approach should work without requiring a true dynamical simulation; that is, it works on static plans.

9.2.1.7. Structured design process

Section 4.3 outlines the process of creating a space plan using the proposed physically based approach. To summarize, it is an iterative process of topological resolution, geometric resolution, and user interaction. In the prototype it is possible to automate the switch from topological resolution to geometric resolution. One area of study is to determine the usefulness of this automation, and to find ways in which this process can be improved. Also, when the user interacts with a plan while the simulation is running, it can automatically switch from geometric resolution back to topological resolution. Another area of study is to determine if this is useful to the user, or if it is more of a distraction.

Alternative processes that relate to current theories of design thinking, such as the paradigm simulated by Akin et al. (1988), might also be studied. A typical strategy for solving design problems is to first solve for the most important requirements, then try to fit in successively less important requirements. This process could be automated to try to mimic observed manual processes. For example, instead of having all adjacencies active from the start, first have only the strong adjacencies active, run the simulation until it reaches equilibrium, turn on the next strongest adjacencies, run the simulation, and so on.

9.2.2. Design process

An obvious and important area of study is how this approach will fit in with current architectural design processes, or potentially change those processes. Does it adequately solve some of the problems of schematic design? What kinds of additional work would be required to transition from the conceptual design phase that the approach appears to aid and later design phases? What about the potential need to transition back to conceptual design if an area of a space plan must be revisited? Is it really compelling to designers when they must use it to solve real problems?

Eastman notes that solids modeling has not and will not serve as the “core representational scheme within architectural CAD” due to the abstractions inherent in architectural presentation drawings and the necessity of solid models to be accurate rather than abstract. He suggests that “another approach involving multiple representations is required.” (Eastman, 1987, p. 139) Space planning methodologies in general are most likely one of the many representations that Eastman speaks of. This applies especially to physically based space planning due to the highly specialized physical properties it requires, which would have little use to other representations.

9.2.3. Strengths and level of importance

Each type of design objective has a strength constant specified for it, which is applied to all instances of that type, as described in section 4.2.1.5. The purpose of the constant is to scale the effect of the design objectives so that objectives with the same level of importance maintain the same relative strength to each other. Strength constants used in the prototype ranged between 1.0 and 500.0, but further study is needed to determine valid values for these constants. For example, the strength of exterior objectives might need to be adjusted relative to adjacency objectives so that

two ‘Important’ objectives, one of each type, have the same expected effect relative to each other.

Each instance of a design objective has a level of importance specified for it, which is applied to all objectives with the same level of importance, as described in section 4.2.1.4. The values for level of importance range between 0.0 and 1.0, and its purpose is to differentiate ‘strong’ vs. ‘weak’ design objectives. Further study is needed to determine how values set by the designer are scaled within this range. For example, it might be found that a difference of 0.2 between two adjacency types might not provide a corresponding ‘expected’ difference in a space plan. An underlying scale, such as a logarithmic scale, might be found to be more appropriate than the linear scale used in the prototype.

9.2.4. Exposure of physically based nature

The design of an application’s user interface is a continual challenge, especially in areas where the application is attempting to introduce new concepts and there is a disconnect between the user’s and the application’s model of the problem. Should the underlying physically based nature of the approach be exposed to the designer?

What would be the effect on the designer if more specific physically based terminology was used? Or is it important to hide this terminology and attempt to use terminology that is more familiar to the designer? What physically based graphic elements are most useful to the designer during the process of design? For example, the display of objective force vectors could be studied to determine if they are in any way useful to designers, such as to help answer the design question “Why is this space where it is?”

9.2.5. Mass

In this prototype, the mass for all space and wall nodes is set to a value of 1.0. These mass values might be an additional factor to consider in modeling certain design objectives. For example, the mass of the center node of a space might be set based on the space's level of importance. A space that is important might have a greater mass than one that is less important, in which case it is less likely to move due to the influence of the less important space.

9.2.6. Effects on design fixation

Does the proposed approach potentially reduce design fixation? The combined automated and manual nature of the approach suggests that a designer can quickly produce a number of designs, save them temporarily using thumbnails, and continually refine more than one to arrive at a suitable design.

9.2.7. Usefulness with different classes of plan types

How does the usefulness/behavior of this approach differ for different 'types' of architectural programs? For example, is there a significant difference between 'tight' programs with many adjacencies between spaces, such as the counseling center listed in appendix C, and 'loose' programs such as the residence listed in appendix D?

9.2.8. Extension to other design domains

The statement made in the Introduction is that 'space planning is a suitable domain in which to develop and test the hypothesis that physically based techniques can be applied to design processes.' Can the results of this research be applied to other design domains? In other words, can the geometric elements in other domains be

modeled with nodes, and can the design objectives as described here be applied to those domains?

9.2.9. Integration with other space planning methods

Section 7.1.6 describes the potential of integrating this approach, with its more useful manipulative abilities, with other approaches that provide more optimized plans. Further work is possible to determine if there is designs are improved when starting with a more optimal plan, or if designer control is ‘good enough’ for most design projects.

9.3. Conclusion

As quoted in Lawson (1997, p. 154), Michael Wilford uses the analogy of a “juggler who’s got six balls in the air . . . and an architect is similarly operating on at least six fronts simultaneously and if you take your eye off one of them and drop it, you’re in trouble.” The physically based space planning approach proposed in this dissertation should allow the designer to keep more things in mind while exploring the design space – in effect it allows the designer to ‘juggle more balls.’ More important, it allows the designer to juggle them better, with more finesse, and more enjoyment.

REFERENCES

- Akin O, Dave B, Pithavadian S, 1988, "Heuristic generation of layouts(HeGeL): Based on a paradigm for problem structuring," in *Artificial Intelligence in Engineering: Design*, Ed J Gero (Elsevier, Amsterdam) pp 413–444
- Alexander C, 1964, *Notes on the Synthesis of Form* (Harvard University Press, Cambridge, Massachusetts)
- Alexander C, 1979, *The Timeless Way of Building* (Oxford University Press, New York)
- Alexander C, Ishikawa S, Silverstein M, 1977, *A Pattern Language: Towns, Buildings, Construction* (Oxford University Press, New York)
- Arnheim R, 1966, "Gestalt psychology and artistic form," in *Aspects of Form*, Ed L L Whyte (Indiana University Press, Bloomington, Indiana) pp 196–208
- Arnheim R, 1969, *Visual Thinking* (University of California Press, Berkeley, California)
- Arnheim R, 1977, *The Dynamics of Architectural Form* (University of California Press, Berkeley, California)
- Balachandran M, Gero J S, 1987, "Dimensioning of architectural floor plans under conflicting objectives," *Environment and Planning B: Planning and Design* **14** 29–37
- Baraff D, 1989, "Analytical methods for dynamic simulation of non-penetrating rigid bodies," in *Computer Graphics, Proceedings of SIGGRAPH 89*, Ed J Lane, ACM SIGGRAPH, Boston, Massachusetts, July 31 - August 4, pp 223–232
- Barzel R, Barr A H, 1988, "A modeling system based on dynamic constraints," in *Computer Graphics, Proceedings of SIGGRAPH 88*, Ed J Dill, ACM SIGGRAPH, Atlanta, Georgia, August 1–5, pp 179–188
- Bohm D, 1995, "Soma-significance: A new notion of the relationship between the physical and the mental," *DynaPsych*, <http://www.goertzel.org/dynapsyc/1995/bohm.html>
- Bohm D, Peat F D, 1987, *Science, Order, and Creativity* (Bantam Books, New York)
- Chomsky N, 2003, *Syntactic structures* (Mouton & Co, The Hague)
- Combs A, Germaine M, Goertzel B, Eds, 2003, *Mind in Time: The Dynamics of Thought, Reality, and Consciousness* (Hampton Press, Cresskill, New Jersey)
- Cooper A, 1999, *The Inmates Are Running the Asylum: Why High-Tech Products Drive Us Crazy and How to Restore the Sanity* (Sams, Indianapolis)

- de Berg M, van Kreveld M, Overmars M, Schwarzkopf O, 1997, *Computational Geometry: Algorithms and Applications* (Springer-Verlag, Berlin)
- Dickson G R, 1959, *Dorsai!* (Ace Science Fiction Books, New York)
- Dietrich E, Markman A B, Eds, 2000, *Cognitive Dynamics: Conceptual and Representational Change in Humans and Machines* (Lawrence Erlbaum Associates, Mahwah, New Jersey)
- Eastman C M, 1987, "Fundamental problems in the development of computer-based architectural design models," in *Computability of Design*, Ed Y E Kalay (John Wiley & Sons, New York) pp 133-140
- Flemming U, 1978, "Wall representations of rectangular dissections and their use in automated space allocation," *Environment and Planning B: Planning and Design* **5**(2) 215-232
- Flemming U, 1987, "More than the sum of parts: The grammar of Queen Anne houses," *Environment and Planning B: Planning and Design* **14** 323-350
- Flemming U, 1989, "More on the representation and generation of loosely-packed arrangements of rectangles," *Environment and Planning B: Planning and Design* **16** 327-359
- Flemming U, Chien S F, 1995, "Schematic layout design in seed environment," *Journal of Architectural Engineering* **1**(4) 162-169
- FLTK, 2003, *Fast Light Tool Kit*, <http://www.fltk.org>
- Focillon H, 1934, *The Life of Forms in Art* (Zone Books, New York), 1992 edition
- Foley J D, van Dam A, Feiner S K, Hughes J F, 1992, *Computer Graphics: Principles and Practice* (Addison-Wesley, Reading, Massachusetts)
- Gero J S, 1998, "Adaptive systems in designing: New analogies from genetics and developmental biology," in *Adaptive Computing in Design and Manufacturing* (Springer, London) pp 3-12
- Gero J S, Kazakov V A, 1998, "Evolving design genes in space layout planning problems," *Artificial Intelligence in Engineering* **12**(3) 163-176
- Gilleard J, 1978, "LAYOUT - a hierarchical computer model for the production of architectural floor plans," *Environment and Planning B: Planning and Design* **5**(2) 233-241
- GLTT, 2001, *OpenGL True Type font library*, <http://gltt.sourceforge.net>
- Grant D P, 1983, "A bibliography on space planning methods classified by type of method," *Design Methods and Theories* **18**(2) 83-92

- Grimsdale R L, Chang C W, 1996, "The layout design language: A technique for generating layout plans," *Computer Graphics Forum* **15**(2) 97–106
- Gross M D, Ervin S, Anderson J, Fleisher A, 1987, "Designing with constraints," in *Computability of Design*, Ed Y E Kalay (John Wiley & Sons, New York) pp 53–68
- Harada M, 1997, *Discrete/Continuous Design Exploration by Direct Manipulation*, Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, Pennsylvania
- Harada M, Witkin A, Baraff D, 1995, "Interactive physically-based manipulation of discrete/continuous models," in *Computer Graphics, Proceedings of SIGGRAPH 95*, Ed R Cook, ACM SIGGRAPH, Los Angeles, California, August 6–11, pp 199–208
- Haumann D, 1987, "Modeling the physical behavior of flexible objects," in *Topics in Physically-Based Modeling; SIGGRAPH Course Notes, Volume 17* (ACM SIGGRAPH, San Francisco, California, July 22–26)
- Haumann D R, Parent R E, 1988, "The behavioral test-bed: Obtaining complex behavior from simple rules," *The Visual Computer* **4** 332–347
- Hawking S, 1988, *A Brief History of Time: From the Big Bang to Black Holes* (Bantam Books, New York)
- Heidegger M, 1962, *Being and Time* (Harper and Row, New York), Translated by John Macquarrie and Edward Robinson
- Heisserman J, 1994, "Generative geometric design," *IEEE Computer Graphics and Applications* **14** 37–45
- House D H, Kocmoud C J, 1998, "Continuous cartogram construction," in *Proceedings of Visualization '98*, Research Triangle Park, North Carolina, October 18–23, pp 197–204
- Jo J H, Gero J S, 1998, "Space layout planning using an evolutionary approach," *Artificial Intelligence in Engineering* **12**(3) 149–162
- Johnson T E, Weinzapfel G, Perkins J, Ju D C, Solo T, Morris D, 1970, "IMAGE: An interactive graphics-based computer system for multi-constrained spatial synthesis," Technical report, Massachusetts Institute of Technology, Department of Architecture, Cambridge, Massachusetts
- Kalay Y E, 1989, *Modeling Objects and Environments* (John Wiley & Sons, New York)
- Karlen M, 1993, *Space Planning Basics* (Van Nostrand Reinhold, New York)
- Kelso J A S, 1995, *Dynamic Patterns: The Self-Organization of Brain and Behavior* (MIT Press, Cambridge, Massachusetts)

- Koutamanis A, 2000, "Digital architectural visualization," *Automation in Construction* **9**(4) 347–360
- Laseau P, 1980, *Graphic Thinking for Architects and Designers* (Van Nostrand Reinhold, New York)
- Lawson B, 1997, *How designers think* (Architectural Press, Oxford), 3rd edition
- Leonov M V, Nikitin A G, 1997, "A closed set of algorithms for performing set operations on polygonal regions in the plane," <http://members.xoom.com/msleonov/pbpaper.html>, link to pbedraft.zip
- Liggett R S, Mitchell W, 1981a, "Interactive graphic floor plan layout method," *Computer-Aided Design* **13**(5) 289–298
- Liggett R S, Mitchell W J, 1981b, "Optimal space planning in practice," *Computer-Aided Design* **13**(5) 277–288
- Mandal C, Qin H, Vemuri B C, 1997, "Dynamic smooth subdivision surfaces for data visualization," in *Proceedings of IEEE Visualization '97*, IEEE, Phoenix, Arizona, October 19–24, pp 371–377
- Martini K, 1995, "Hierarchical geometric constraints for building design," *Computer-Aided Design* **27**(3) 181–91
- McCullough M, 1996, *Abstracting Craft: The Practiced Digital Hand* (MIT Press, Cambridge, Massachusetts)
- McKim R H, 1972, *Experiences in Visual Thinking* (Wadsworth Publishing Co., Belmont, California)
- Merriam-Webster's Collegiate Dictionary*, 1995 (Merriam-Webster, Springfield, Massachusetts), 10th edition
- Mitchell W J, 1977, *Computer-Aided Architectural Design* (Petrocelli, New York)
- Mitchell W J, McCullough M, 1995, *Digital Design Media* (Van Nostrand Reinhold, New York)
- Moore M, Wilhelms J, 1988, "Collision detection and response for computer animation," in *Computer Graphics, Proceedings of SIGGRAPH 88*, Ed J Dill, ACM SIGGRAPH, Atlanta, Georgia, August 1–5, pp 289–298
- Murray J D, vanRyper W, 1994, *Encyclopedia of Graphics File Formats* (O'Reilly & Associates, Sebastopol, California)
- Negroponte N, 1975, *Soft architecture machines*. (MIT Press, Cambridge, Massachusetts)
- Newell A, Simon H, 1976, "Computer science as empirical enquiry: Symbols and search," *Communications of the Association for Computing Machinery* **19** 113–126

- Papper M, Danahy J, Baecker R, 1991, "Predictable modelling interaction using high-level constraints: Making objects behave as they would in our environment," in *Reality and Virtual Reality; Proceedings of the 1991 Conference of the Association for Computer-Aided Design in Architecture*, Eds G Goldman, M S Zdepski, University of California, Los Angeles, pp 211–22
- Peterson I, 2000, "Changes of mathematical state: Untangling a web of conflicting demands can be tough on computers," *Science News* **157**(19) 296–297, 302
- Pfefferkorn C E, 1975, "The design problem solver: A system for designing equipment and furniture layouts," in *Spatial Synthesis in Computer-Aided Building Design*, Ed C M Eastman (John Wiley & Sons, New York) pp 98–146
- Pinker S, 1997, *How the Mind Works* (W. W. Norton & Co., New York)
- PNG, 2002, PNG Reference Library, <http://www.libpng.org/pub/png>
- Port R F, van Gelder T, Eds, 1995, *Mind as Motion : Explorations in the Dynamics of Cognition* (MIT Press, Cambridge, Massachusetts)
- Press W H, Teukolsky S A, Vetterling W T, Flannery B P, 1992, *Numerical Recipes in C: The Art of Scientific Computing* (The Press Syndicate of the University of Cambridge, Cambridge, England)
- Prusinkiewicz P, Lindenmayer A, 1990, *The Algorithmic Beauty of Plants* (Springer-Verlag, New York)
- Qin H, 1998, "A physics-based geometric modeling and design system," <http://www.nsf.gov/cgi-bin/showaward?award=9896170>
- Qin H, Mandal C, Vemuri B C, 1998, "Dynamic catmull-clark subdivision surfaces," *IEEE Transactions on Visualization and Computer Graphics* **4**(3) 215–229
- Qin H, Terzopoulos D, 1995, "Dynamic NURBS swung surfaces for physics-based shape design," *Computer-Aided Design* **27**(2) 111–127
- Reeves W T, 1983, "Particle systems: A technique for modeling a class of fuzzy objects," *ACM Transactions on Graphics* **2**(2) 91–108
- Reynolds C, 1987, "Flocks, herds, and schools: A distributed behavioral model," in *Computer Graphics, Proceedings of SIGGRAPH 87*, Ed M Stone, ACM SIGGRAPH, Anaheim, California, July 27–31, pp 17–24
- Rollo J, 1995, "Triangle and T-square: The windows of Frank Lloyd Wright," *Environment and Planning B: Planning and Design* **22** 75–92
- Rowe P G, 1987, *Design thinking* (MIT Press, Cambridge, Massachusetts)
- Russell S, Norvig P, 1995, *Artificial Intelligence: A Modern Approach* (Prentice-Hall, Englewood Cliffs, New Jersey)

- Sanoff H, 1977, *Methods of Architectural Programming* (Dowden, Hutchinson & Ross, Stroudsburg, Pennsylvania)
- Schön D A, 1983, *The Reflective Practitioner: How Professionals Think in Action* (Basic Books, New York)
- Schutte K, 1995, "An edge labeling approach to concave polygon clipping," <http://www.ph.tn.tudelft.nl/~klamer/clip.ps.gz>
- Scriabin M, Vergin R C, 1975, "Comparison of computer algorithms and visual based methods for plant layout," *Management Science* **22**(2) 172–187
- Sims K, 1991, "Artificial evolution for computer graphics," in *Computer Graphics, Proceedings of SIGGRAPH 91*, Ed T W Seegerberg, ACM SIGGRAPH, Las Vegas, Nevada, July 28 - August 2, pp 319–328
- Sims K, 1994, "Evolving virtual creatures," in *Computer Graphics, Proceedings of SIGGRAPH 94*, Ed A S Glassner, ACM SIGGRAPH, Orlando, Florida, July 24–29, pp 15–22
- Smithers T, 1994, "Exploration in design: Discussion," in *Formal Design Methods in CAD*, Eds J S Gero, E Tyugu (Elsevier, Amsterdam) pp 337–350
- Stiny G, Gips J, 1972, "Shape grammars and the generative specification of painting and sculpture," in *Information Processing 71*, Ed C V Freiman (North-Holland, Amsterdam) pp 1460–1465
- Stiny G, Mitchell W J, 1978, "The Palladian grammar," *Environment and Planning B: Planning and Design* **5** 5–18
- Sutherland I E, 1963, *Sketchpad, A man-machine graphical communication system*, Ph.D. dissertation, Massachusetts Institute of Technology, <http://www.cl.cam.ac.uk/TechReports/UCAM-CL-TR-574.pdf>
- Terzopoulos D, Platt J, Barr A H, Fleischer K, 1987, "Elastically deformable models," in *Computer Graphics, Proceedings of SIGGRAPH 87*, Ed M Stone, ACM SIGGRAPH, Anaheim, California, July 27–31, pp 205–214
- Thompson D W, 1992, *On Growth and Form: The Complete Revised Edition* (Dover Publications, Inc., New York)
- Tobin K L, 1991, "Constraint-based three-dimensional modeling as a design tool," in *Reality and Virtual Reality; Proceedings of the 1991 Conference of the Association for Computer-Aided Design in Architecture*, Eds G Goldman, M S Zdepski, University of California, Los Angeles, pp 193–209
- van Gelder T J, 1998, "The dynamical hypothesis in cognitive science," *Behavioral and Brain Sciences* **21** 1–14
- van Gelder T J, 1999, "Revisiting the dynamical hypothesis," <http://www.arts.unimelb.edu.au/~tgelder/papers/DDH.pdf>

- Vatti B R, 1992, "A generic solution to polygon clipping," *Communications of the ACM* **35**(7) 56–63
- Ware C, 2000, *Information Visualization: Perception for Design* (Morgan Kaufmann, San Francisco)
- Weinzapfel G, Handel S, 1975, "IMAGE: Computer assistant for architectural design," in *Spatial Synthesis in Computer-Aided Building Design*, Ed C M Eastman (John Wiley & Sons, New York) pp 61–97
- Whited T, 1980, "An improved illumination model for shaded display," *Communications of the ACM* **23**(6) 343–349
- Wilhelm M R, Ward T L, 1987, "Solving quadratic assignment problems by simulated annealing," *IIE Transactions* **19**(1) 107–119
- Williams J E, Trinklein F E, Metcalfe H C, 1976, *Modern Physics* (Holt, Rinehart and Winston, New York)
- Witkin A, Baraff D, Eds, 1997, *Physically Based Modeling: Principles and Practice; SIGGRAPH Course Notes, Volume 19* (ACM SIGGRAPH, Los Angeles, California, August 3–8)
- Witkin A, Gleicher M, Welch W, 1990, "Interactive dynamics," *Computer Graphics* **24**(2) 11–21
- Witkin A, Kass M, 1988, "Spacetime constraints," in *Computer Graphics, Proceedings of SIGGRAPH 88*, Ed J Dill, ACM SIGGRAPH, Atlanta, Georgia, August 1–5, pp 159–168
- Witkin A, Welch W, 1990, "Fast animation and control of nonrigid structures," in *Computer Graphics, Proceedings of SIGGRAPH 90*, Ed F Baskett, ACM SIGGRAPH, Dallas, Texas, August 6–10, pp 243–252
- Woodbury R F, 1987, "Strategies for interactive design systems," in *Computability of Design*, Ed Y E Kalay (John Wiley & Sons, New York) pp 11–36
- Yoon K B, 1992, *A Constraint Model of Space Planning* (Computational Mechanics Publications, Southampton, United Kingdom)
- Zalik B, Gombosi M, Podgorelec D, 1998, "A quick intersection algorithm for arbitrary polygons," in *14th Spring Conference on Computer Graphics and its Applications*, Ed L Szirmay-Kalos, Budmerice, Slovakia, 23–25 April, pp 195–204, <http://www.dcs.fmph.uniba.sk/~sccg/proceedings/1998>

APPENDIX A

PROTOTYPE INTERFACE

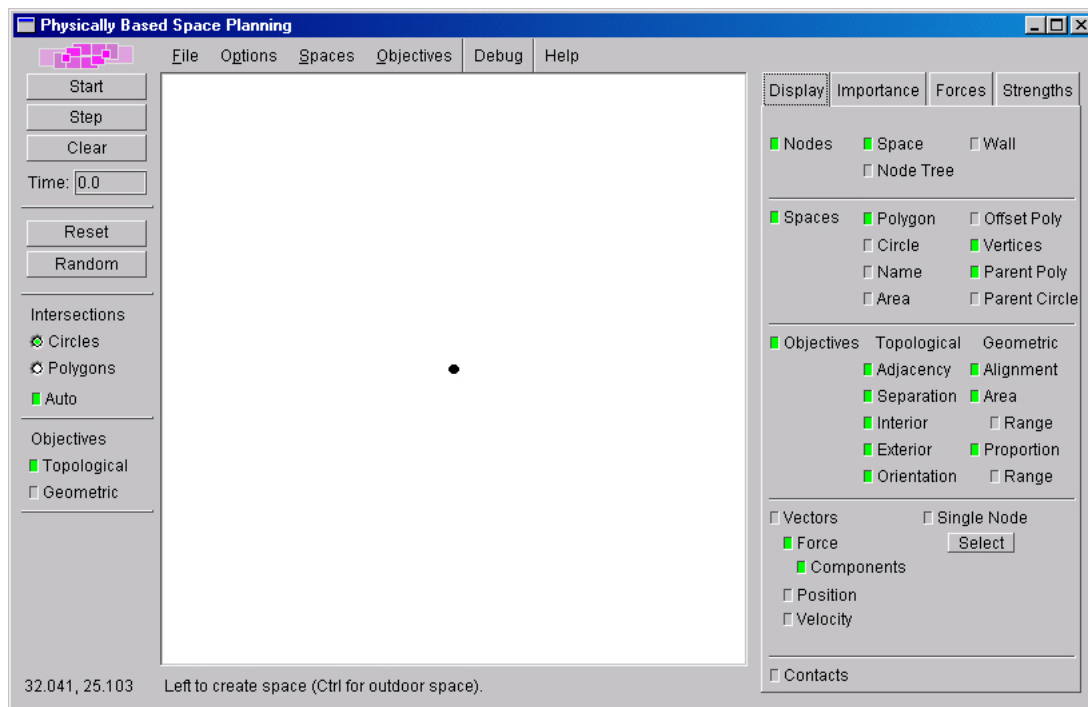


Fig. A-0. Physically based space planning prototype: Main interface

Physically Base Space Planning, shown in figure A-0, is a prototype application intended to produce architectural space plans. The main objects that are created and modified are Spaces, which represent individual physical volumes in or around a building, and Design Objectives, which represent the intentions of the designer regarding the location of spaces relative to each other and the dimensions of space boundaries.

The purpose of this prototype is to demonstrate the feasibility of an approach to space planning that models the elements of a space plan as physical masses and the intentions of the designer as physical forces that act upon those masses.

Since this is a prototype developed for research purposes, many of the controls that are available would not normally be available in a commercial or ‘working’ application. They exist to support the development of either research ideas or the software itself.

The main areas of the initial application window are:

Drawing Area. The drawing area is the main control and is located in the middle of the main application window. All object display and direct user interaction takes place in the drawing area. Directions to the user as well as potential actions based on the current cursor position are displayed in the area immediately below the drawing area. Drawing area graphic elements and interaction are described in section A.1.4.

Menus. The menu bar is in the standard location along the top of the application window. It makes available many commands, options, and system information. Menu items are described in section A.2.

Simulation Controls. The simulation controls are located on the left of the drawing area and provide the means to control various aspects of the dynamic simulation environment, such as starting, stopping or resetting the simulation. Simulation controls are described in section A.3.

Display Tabs. The display tabs are located on the right of the drawing area and provide the means to control the display of design elements and objectives, as well as control the strength of design objectives. The different display tabs are described in section A.4.

A.1. Drawing area

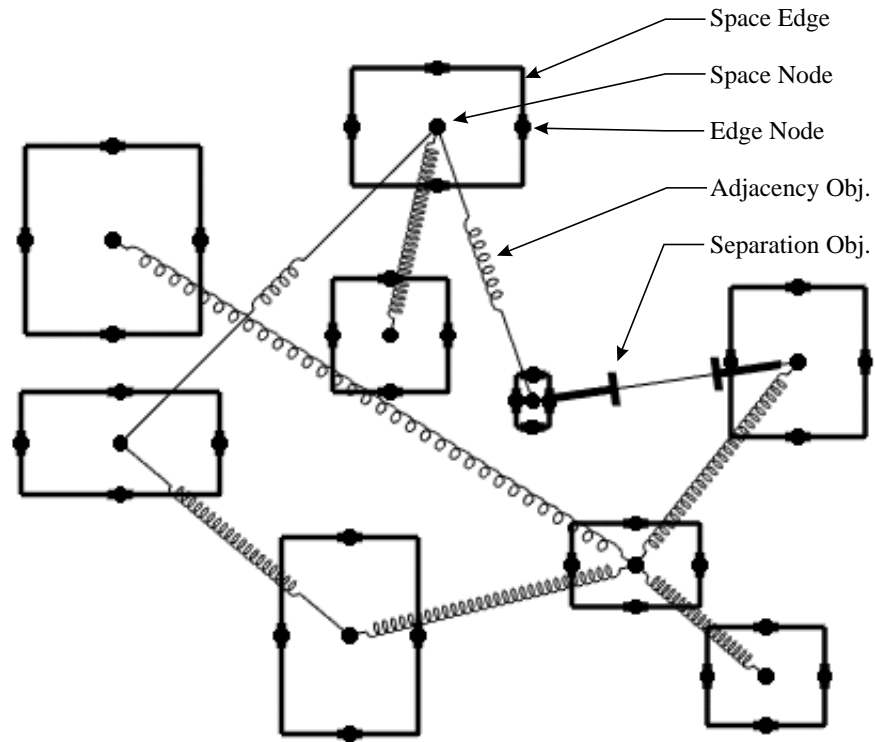


Fig. A-1. Drawing area elements

This section describes all graphic objects that can be drawn in the drawing area. Figure A-1 shows some of the more common elements.

The elements in the drawing area can be displayed with individual colors on a black background, or as black on a white background. The colors noted are those used with the black background.

A.1.1. Space elements

A Space element represents an individual physical volume in or around a building, and is modeled internally as masses for the space itself and for the wall surrounding it. A number of graphic elements are related to features of a single space object or a collection of spaces. When a space is created it is assigned a random color, which most of the individual elements also use. The following elements apply to a single space and are shown in figure A-2.

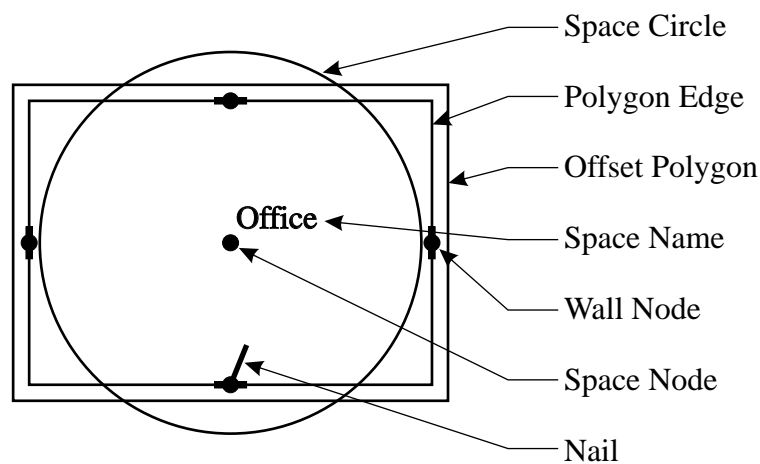


Fig. A-2. Drawing area space elements

Space Node. A Space Node is a space's center node, as described in section 4.1.3. It is typically a point node, which is represented as a dot. Its color is the space's color.

Wall Node. A Wall Node is a node for each edge that defines a space's polygonal boundary, as described in section 4.1.2. It is always a line node, which is represented as a dot with a bar through it parallel to the edge. Its color is the

space's color.

Polygon Edge. A Polygon Edge is the edge of the polygon defining each Space, drawn through each edge's associated Wall Node. Its color is the Space's color.

Space Circle. A Space Circle is the circular representation of the area of a space's polygon. It is used to visualize the internal representation of a space that is active during topological resolution. Its color is the space's color.

Name. A Name is text displaying a space's name. Its color is white.

Offset Polygon. An Offset Polygon is a space's polygon offset to the outside by a predefined value representing a wall thickness. The collection of all offset polygons for a given hierarchical level is used in the polygon union algorithm to find the exterior boundary of a set of spaces (see appendix E). Its color is the space's color.

Nail. A Nail is a graphic indication that a space or wall Node cannot move due to forces applied to it. Its color is white.

Vertices. Vertices, not shown, are small dots that are displayed at the intersections of each pair of coincident edges of a space's polygon. Their color is white.

Outdoor Space. An Outdoor Space, not shown, has all the properties of normal spaces as far as space planning is concerned, except that it does not contribute to the parent polygon (described below) used to represent the exterior face of a building. It is drawn with dotted lines.

The following elements apply to multiple spaces and are shown in figure A-3.

Node Tree. A Node Tree is a way to visualize the hierarchical connections of all nodes in a plan. Shaded white lines are drawn between each parent-child relationship for space nodes, and between each space node and its associated wall nodes. The node tree lines are dark at the parent node and light at the child node.

Parent Polygon. The Parent Polygon is the union of the collection of offset polygons for a given hierarchical level. Its color is yellow.

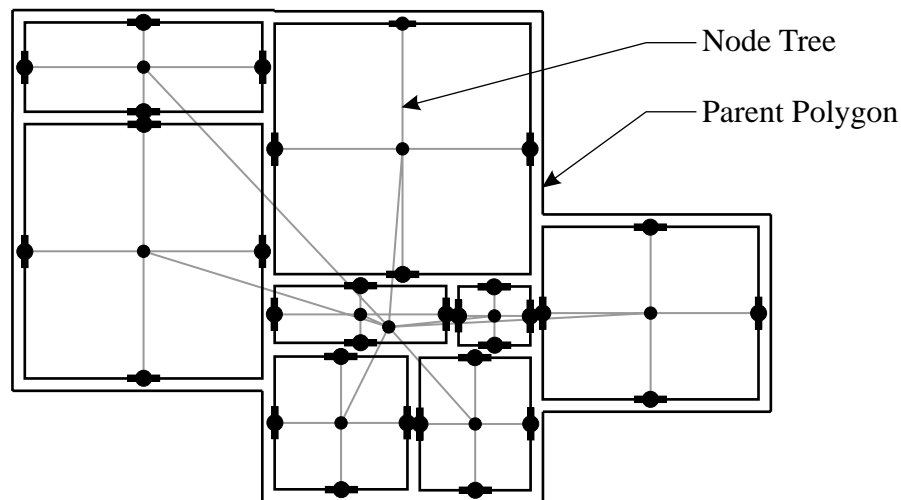


Fig. A-3. Drawing area space elements (cont.)

Parent Circle. The Parent Circle is the smallest circle that can enclose the space center nodes for a given hierarchical level. Also drawn with it are two small circles, one at the **geometric center** and the other at the **center of mass** of the enclosing circle. Its color is gray.

A.1.2. Design objective elements

A Design Objective represents the intentions of the designer regarding the location of spaces relative to each other and the dimensions of space boundaries. It is modeled internally as an object that applies forces to space or wall masses (nodes). Some individual design objectives are shown in figure A-4.

Unless otherwise noted, the color of a design objective is defined by its level of importance (see section A.4.2).

Adjacency. An Adjacency Objective can be applied to two spaces with the intent that they be located next to each other. It is drawn with a thin line connecting the center nodes of the two spaces, overlaid with either a thick line or a spring, whose

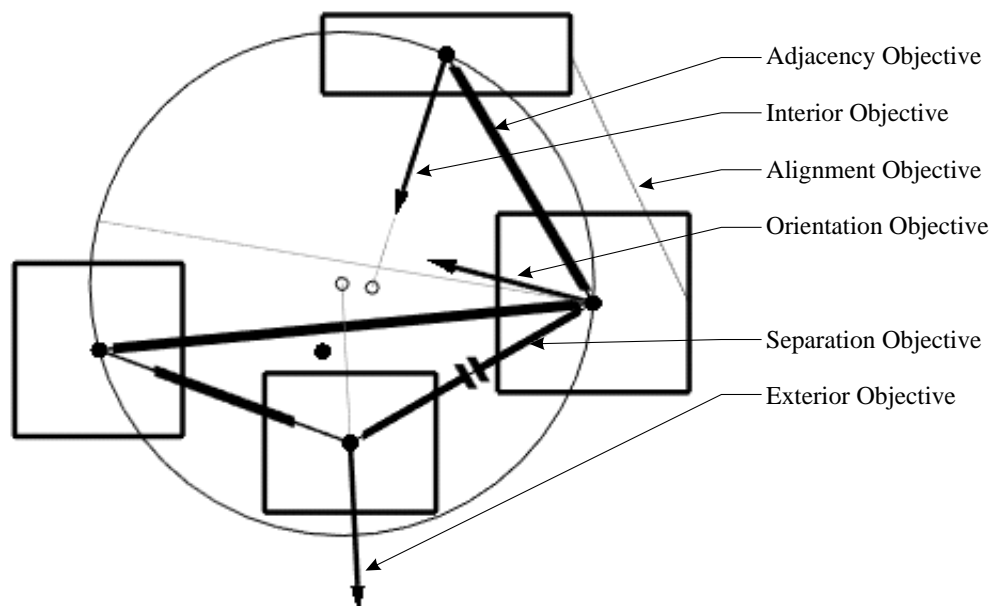


Fig. A-4. Drawing area objective elements

length represents its level of importance.

Separation. A Separation Objective can be applied to two spaces with the intent that they be located apart from each other. It is drawn with a thin line connecting the center nodes of the two spaces, overlaid with two ‘T’ shaped thick lines whose length represents its level of importance.

Interior. An Interior Objective can be applied to a single space with the intent that it be located at the center of a set of spaces. It is drawn as an arrow starting at the space’s center node, pointing toward the center of mass of the space’s parent circle. (See Parent Circle in section A.1.1)

Exterior. An Exterior Objective can be applied to a single space with the intent that it be located on the outside of a set of spaces. It is drawn as an arrow starting at the space’s center node, pointing away from its parent circle’s geometric center. (See Parent Circle in section A.1.1)

Orientation. An Orientation Objective can be applied to a single space with the intent that it be located on the outside of a set of spaces, similar to an exterior objective, but in a specified direction or orientation relative to the center of that set. It is drawn as an arrow starting at the space's center node, pointing toward a location on the space's parent circle, which is found by projecting the specified direction vector from the parent circle's geometric center. (See Parent Circle in section A.1.1)

Area. An Area Objective is implicitly created when a space is created, with the intent that the space maintains a specified area. It is drawn as a shaded polygon whose color depends on the difference between the actual area and the intended area. It is black if the actual area equals the intended area, red if the actual area is smaller, and blue if it is larger. The intensity (value in the HSV color space (Foley et al., 1992)) of the color indicates the degree of difference. It is also drawn with polygons representing the target area and minimum and maximum ranges.

Proportion. A Proportion Objective is implicitly created when a space is created, with the intent that the space maintains a specified proportion. It is drawn with polygons representing the intended proportion and minimum and maximum ranges.

Alignment. An Alignment Objective can be applied to two walls with the intent that their nodes either align with each other or are offset by a specified amount. If the alignment is currently not met it is drawn with arrows whose start points are at each wall node, whose length is the distance required for each node to reach alignment, and whose direction is toward the point where each node would reach alignment. If the alignment is currently met, it is drawn as a dot. In addition, dotted lines are drawn between each node indicating the offset distance and the distance of 'misalignment.'

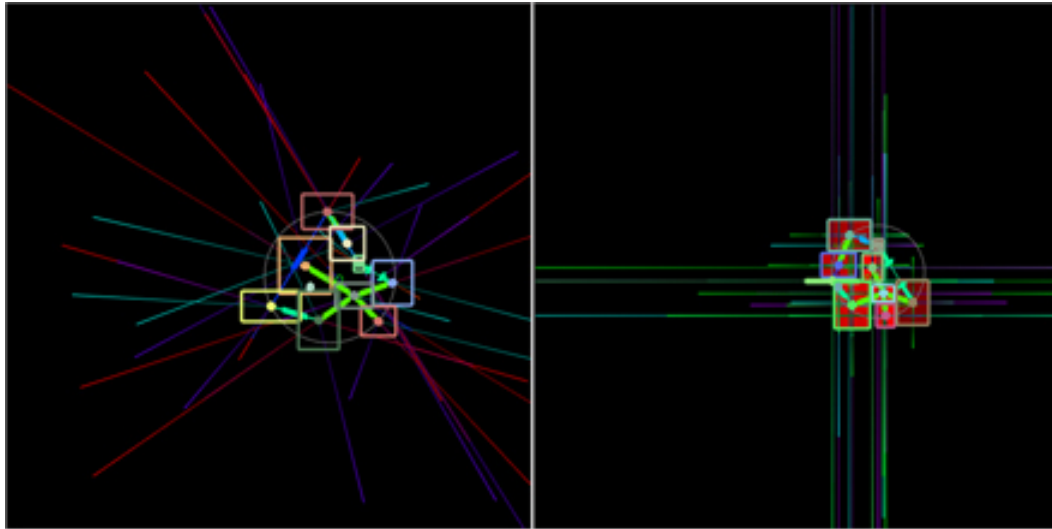


Fig. A-5. Drawing area force elements on all nodes

Force Vectors. Every design objective, not just those that have their own graphic representation, can apply a number of forces to the space or wall nodes of a space plan. Each of these individual forces can be drawn as a colored line whose length corresponds to the magnitude of the force, starting from the node to which the force is applied and drawn in the direction that the force is applied. The color for different types of forces can be defined individually. (see section A.4.3) It is possible to display forces on all nodes, as shown in figure A-5, or on a single selected node, as shown in figure A-6.

Instead of drawing each individual force on a node, the sum of all forces applied to a node can be drawn. This total force is drawn as a green line, similar to the display of individual forces just described.

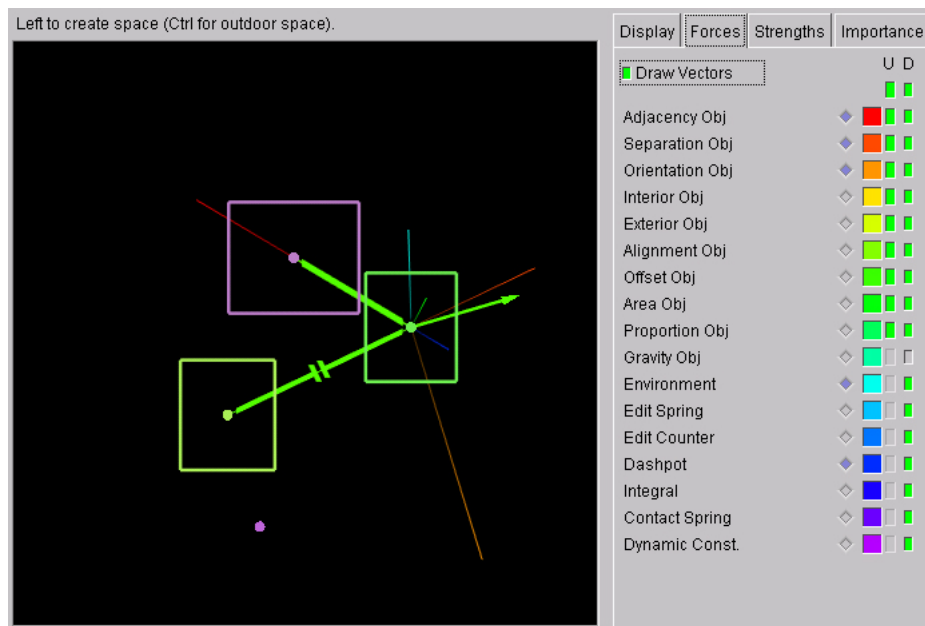


Fig. A-6. Drawing area force elements on a single node

A.1.3. Miscellaneous elements

There are a few more graphic elements that can be displayed in the drawing area, in addition to those representing spaces and design objectives.

Collision. A Collision indicator is a small green dot displayed when two spaces are in contact with each other, with lines drawn from each space's center node to the collision dot. Collision display only occurs during geometric resolution, when the collision detection method is via polygons and not circles. This element was used during prototype development to debug the dynamic constraints used to maintain the appropriate distance between spaces. (see section 4.3.2)

Position Vector. A Position Vector displays the change in position of a node, and is a blue line drawn from the node's last position to its current position.

Velocity Vector. A Velocity Vector displays the current node's velocity, and

Table A-0. Kinetic energy graph colors

Equilibrium	Black Background	White Background
No	Red	Gray
Yes	Green	Black

is a cyan line with a length equal to the magnitude of the velocity, drawn from the node's current position in the direction of the velocity.

Coordinate Axes. The Coordinate Axes display the X, Y, and Z axes of the world coordinate system, colored red, green, and blue, respectively. (XYZ = RGB).

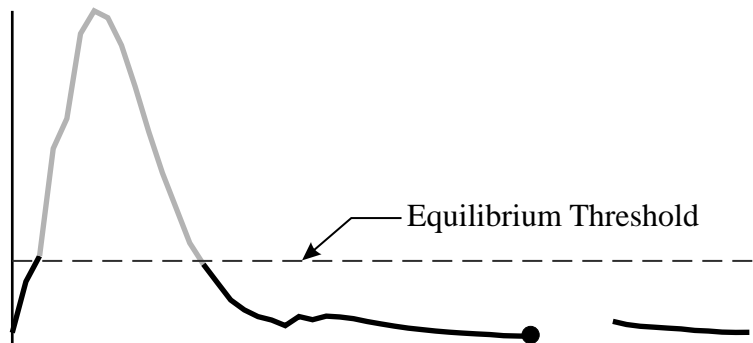


Fig. A-7. Drawing area kinetic energy graph

Kinetic Energy Graph. The Kinetic Energy Graph, as shown in figure A-7, is a display of the kinetic energy in the simulation over a period of time. This energy measurement is used to determine whether or not the system is in equilibrium, which is used in turn to automatically change the intersection method from circles to polygons, as discussed in section 4.3. See table A-0 for the colors used to draw the graph.

Table A-1. Basic drawing area mouse interaction

Left Mouse	Create and Modify Spaces
Right Mouse	Create and Modify Design Objectives
Middle Mouse	Change View
Shift Key	Constrain in some way
Control Key	Miscellaneous option, or customize (unconstrain)

A.1.4. Interaction

The user can interact with the elements displayed in the drawing area with various combinations of mouse movements, mouse button clicks, and modifier key options. Table A-1 shows a basic summary or ‘philosophy’ of mouse and keyboard interaction in the drawing area, and table A-2 shows a complete outline of mouse interaction in the drawing area for the combinations of mouse-key buttons and drawing area elements, which are further described below.

The line of text immediately below the drawing area displays the user interaction required in the drawing area when a specific command is active, as well as potential user interaction when no command is active and the cursor hovers over different elements.

Most of the interaction required in the drawing area can be done in a modeless manner; that is, most commands used to create and modify objects can be implicitly activated based on what is currently under the cursor and do not have to be explicitly activated by selecting buttons or menu items. For example, if the user clicks and drags a space center node, the space is moved; if a space edge, the space is reshaped. The discussion below is presented in hierarchical order; for example, if a space node and a space edge are under the cursor position when the left mouse button is activated, the command associated with the space node is activated.

Unless otherwise noted, ‘click’ or ‘double-click’ indicates clicking with the left

Table A-2. Outline of drawing area mouse interaction

	Empty	Spaces			Objectives		
		Node	Edge	Vertex	All Topo	Orientation	Alignment
Left	Create space	Move	Resize	Resize			
Left-Shift	Constrain to square	Constrain to axes		Constrain proportion			
Left-Control	Create outdoor space			Constrain area			
Right	Display coords	Create adjacency	Create alignment		Modify strength, constrained	Modify direction	Modify offset
Right-Shift						Modify direction, constrained	Modify offset, constrained
Right-Control		Create separation			Modify strength, custom		
Middle	Pan						
Middle Wheel	Zoom						
Middle-Shift	Zoom						
Middle-Control	Rotate (only in perspective view)						

mouse button.

Space Node. To move a space, click on the center node of the space, drag, and unclick to select its new location. The behavior of the space while moving and its resulting position depends on the currently selected edit method (see section A.2.2). Pressing the Shift key while dragging will constrain the node position to a horizontal or vertical direction relative to its position when initially selected.

To add an adjacency objective between two spaces, right-click over one space node, drag, and unclick over another space node.

To add an separation objective between two spaces, ctrl-right-click over one space node, drag, and unclick over another space node.

Space Edge. To resize a space polygon, click on any space edge, drag, and unclick. Edge movement is automatically constrained to maintain a rectangle. The

space's area and proportion objectives will be updated to the new polygon.

To add an alignment objective between two space edges, click on one space edge, drag, and unclick on a space edge from another space. During the dragging process a gray dashed line will be drawn between the previously selected edge and the current cursor position. If two edges from different spaces are successfully selected, a red line will be drawn between the centers of the edges. If the two edges are parallel the alignment objective will attempt to make them co-linear. If they are perpendicular the alignment objective will attempt to align one edge with the center of the other.

Space Vertices. To resize a rectangular space polygon, click on any vertex, drag, and unclick. With no key modifier pressed, the change in shape is unconstrained, and the space's area and proportion objectives will be updated to the new rectangle. Pressing the Shift key will constrain the rectangle's proportions, changing and updating the space's area objective. Pressing the Control key will constrain the rectangle's area, changing and updating the space's proportion objective.

Space Name. To move a space name, click, drag, and unclick. To change it, double-click over the name and type the new name in the input box displayed near it, and enter to change it or press escape to leave it unchanged.

All Topological Objectives. To change the strength of any topological objective, click on it, drag, and unclick. During dragging the current cursor location is projected perpendicularly to the line representing the objective, and that point is used to set the new strength. By default that point will be 'snapped' to the nearest level of importance defined on the Importance tab (see section A.4.2). To customize the strength or set an unconstrained level of importance, press the control key while dragging.

Orientation Objective. Orientation objectives have other types of interaction in addition to those provided above. To change the direction of an orientation

objective, shift-right-click on an orientation objective, drag, and unclick. During dragging a gray dashed line will be drawn from the space node in the direction of the current cursor location. While the shift key is pressed and the right mouse button is down, the direction will be constrained to 45° increments; with the shift key up the direction will be unconstrained.

Alignment Objective. Alignment objectives have other types of interaction in addition to those provided above. To change the offset value of an alignment objective, right-click on an alignment objective, drag, and unclick. To constrain the offset to the nearest foot, hold the Shift key while dragging. To set the offset directly, double click on it with the left mouse button.

Empty area. To create a new rectangular space object, click when the cursor is over an empty area of the drawing area to set one corner of the space, drag, and unclick to set the opposite corner. The new space will have an area objective and a proportion objective defined by the resulting rectangle. Pressing the Shift key while dragging will constrain the shape of the Space to a square. To create an outdoor space, press the Control key before selecting the first corner of the space.

To display the current cursor coordinates, right click. As long as the right mouse button is down the coordinates will be displayed to the left of the Drawing Area below the simulation controls.

Pan view. Pan the current view in the drawing area by clicking and dragging with the middle mouse button.

Zoom view. Zoom the current view in the drawing area by scrolling with the mouse wheel, or clicking and dragging with the middle mouse button while pressing the shift key (dragging up will zoom in, dragging down will zoom out).

Rotate view. When the Camera is set to Perspective mode, rotate the current view in the drawing area by clicking and dragging with the middle mouse button

while pressing the control key.

A.2. Menus

The menu bar provides access to commands and options not available via the controls on the tabs or via direct interaction in the drawing area.

<u>N</u> ew	Ctrl+n
<u>O</u> pen ...	Ctrl+o
<u>S</u> ave	Ctrl+s
Save <u>A</u> s ...	Shift+Ctrl+s
Export DXF ...	
<u>Q</u> uit	Ctrl+q

Fig. A-8. File menu

A.2.1. File menu

The File menu, shown in figure A-8, contains commands for reading and writing files.

The file format created for the Physically Based Space Planning program is called an .apf file, for Architectural Programming File, and is described in detail in appendix B. An .apf file defines a hierarchy of spaces, the adjacency requirements between spaces, and other information affecting the planning of spaces. It was designed as a means of defining an architectural space plan from an architectural program, and to be relatively easy to implement.

New. The New menu item prompts the user to save the current plan if it has changed, then clears the current space plan from the drawing area.

Open. The Open menu item prompts the user to save the current plan if it has

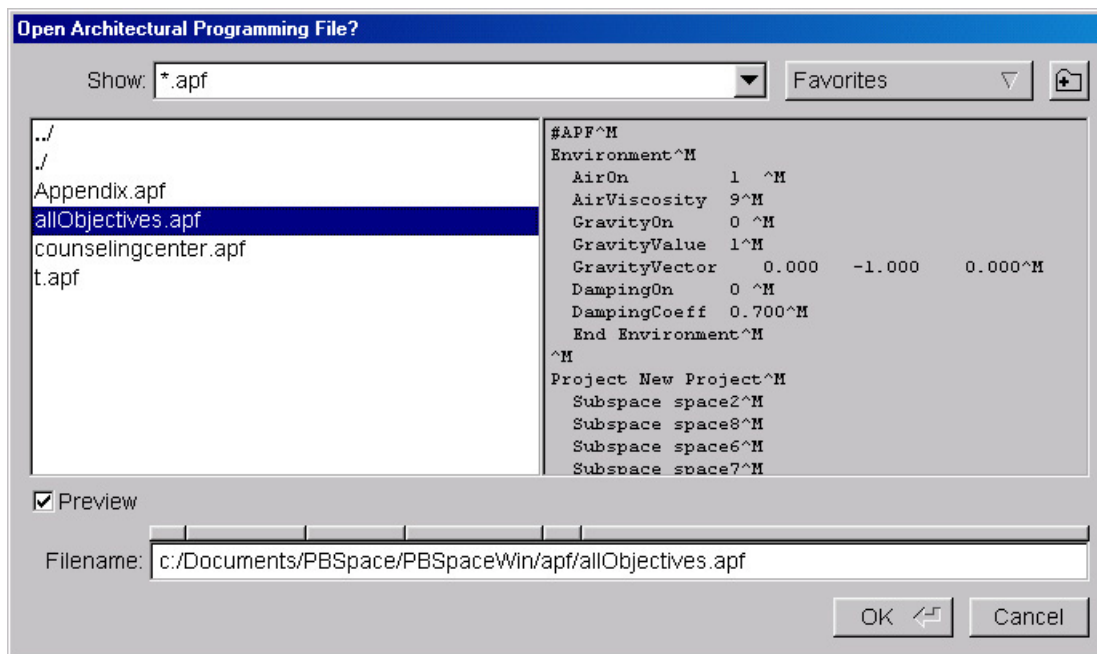


Fig. A-9. File Open dialog

changed, displays a file dialog as shown in figure A-9, and opens a selected .apf file to be displayed in the drawing area.

Save. The Save menu item saves the current plan to an .apf file. If there is no current filename, the file dialog is displayed to select one.

Save As. The Save As menu item displays the file dialog, and saves the current plan to the selected .apf file.

Export DXF. The Export DXF menu item displays a file dialog to select a .dxf file, and exports the current space plan in .dxf format, the Drawing Exchange Format specified by Autodesk Incorporated (Murray and vanRyper, 1994). This option was used primarily to create figures for presentation purposes, and was also intended to provide a means to bring a space plan into a commercial CAD package for further refinement.

Quit. The Quit menu item prompts the user to save the current plan if it has changed, and exits the program.

A.2.2. Options menu

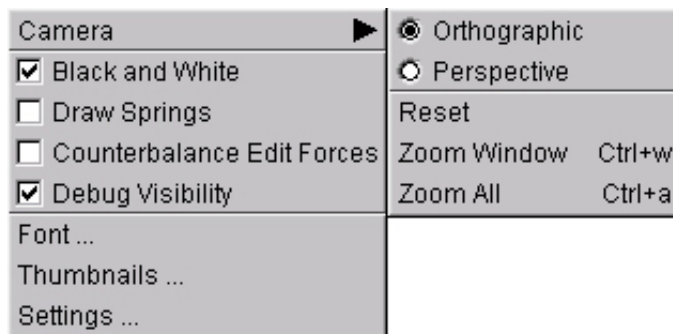


Fig. A-10. Options menu

The Options menu, shown in figure A-10, provides access to setting a large number of miscellaneous options.

Camera. The Camera flyout menu controls the view direction in the Drawing Area. The radio buttons set the current projection method to either **Orthographic** or **Perspective**. The **Reset** item resets the current viewing transformation to the default view for the current projection method. The **Zoom Window** item is used to set the current view to a specified rectangle. The **Zoom All** item is used to set the current view to encompass all spaces.

Black and White. The Black and White menu option toggles the display colors of the drawing area. If it is selected, drawing elements are displayed as black on a white background, otherwise they are displayed in multiple colors on a black background.

Draw Springs. The Draw Springs menu option toggles the display of diagrammatic springs for adjacency objectives. When not set, adjacency objectives are displayed as lines.

Counterbalance Edit Forces. The Counterbalance Edit Forces menu option toggles the application of forces to counterbalance those applied during editing, and also depends on the current edit method specified in the Settings dialog window described in section A.5.3. If this setting is on, for any force applied to a node during editing, such as when a space node is moved, additional forces are applied to all other space nodes such that their sum has equal magnitude but opposite direction to the edit force.

Debug Visibility. The Debug Visibility menu option toggles the display of the Debug menu and other controls used strictly for debugging.

Font ... The Font menu item displays the Font dialog window, which is used to set the font for displaying space names. See section A.5.1.

Thumbnails ... The Thumbnails menu item displays the Thumbnails dialog window, which is used to easily save and restore a number of plans during their conceptual development. See section A.5.2.

Settings ... The Settings menu item displays the Settings dialog window, which is used to control some miscellaneous settings, most controlling the physically based simulation. See section A.5.3.

A.2.3. Spaces menu

The Space menu, shown in figure A-11, contains items used to create and delete spaces.

Space. The Space menu item is used to create a rectangular space element in the drawing area. It is an alternative to the more useful direct interaction method



Fig. A-11. Spaces menu

described in section A.1.4. After selecting the item, click on a point in the drawing area to set one corner of the rectangle, drag, and click on another point to set the opposite corner.

Delete Space. The Delete Space item is used to delete a space element in the drawing area. After selecting the item, click on either a space node or a space edge to delete a space.

Nail. The Nail menu item is used to anchor or ‘nail’ a space node so that it cannot move during the physical simulation. After selecting the item, select a space node.

Unnail. The Unnail menu item is used to unanchor or ‘unnail’ a space node that is currently nailed so that it is free to move. After selecting the item, select a space node.

A.2.4. Objectives menu

The Objectives menu, shown in figure A-12, contains commands for creating a number of the Design Objectives described in section 4.2.

A.2.4.1. Topological Objectives

Adjacency. The Adjacency menu item is used to create an adjacency objective (section 4.2.5.1). After selecting this item, select two space nodes. Adjacency objectives can also be added directly in the drawing area, as described in section A.1.4.

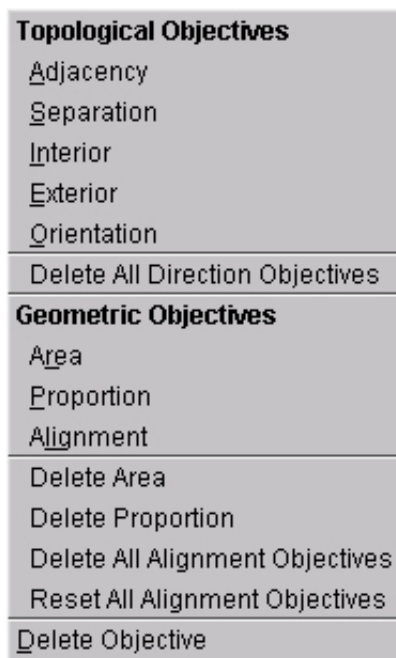


Fig. A-12. Objectives menu

Separation. The Separation menu item is used to create a separation objective (section 4.2.5.2). After selecting this item, select two space nodes. Separation objectives can also be added directly in the drawing area, as described in section A.1.4.

Interior. The Interior menu item is used to apply an interior objective (section 4.2.4.1) to a space. After selecting this menu item, click on a space node. A green line is drawn from the space node pointing to the center of all its sibling spaces.

Exterior. The Exterior menu item is used to apply an exterior objective (section 4.2.4.2) to a space. After selecting this menu item, click on a space node. A green line is drawn from the space node pointing away from the center of all its sibling spaces.

Orientation. The Orientation menu item is used to apply an orientation objective (section 4.2.4.3) to a space. After selecting this menu item, click on a space

node, and then drag in any direction. A red line is drawn from the space node pointing in the direction specified. If the Shift key is pressed while dragging the direction is constrained to 45° increments.

Delete All Direction Objectives. The Delete All Direction Objectives menu item is used to delete all interior, exterior, and orientation objectives in the current plan.

A.2.4.2. Geometric Objectives

Area. The Area Objective menu item is used to create an area objective for a single space. After selecting this item, select a space. Note that area objectives are automatically added to spaces when they are created. This command will only have an effect if an area objective is explicitly deleted on a space.

Proportion. The Proportion Objective menu item is used to create a proportion objective for a single space. After selecting this item, select a space. Note that proportion objectives are automatically added to spaces when they are created. This command will only have an effect if a proportion objective is explicitly deleted on a space.

Alignment. The Alignment Objective menu item is used to create an alignment objective between two space edges. After selecting this item, select the edges from two different spaces. Alignment objectives can also be added directly in the drawing area (see section A.1.4).

Delete Area. The Delete Area menu item is used to delete an area objective from a space. After selecting this item, select a space node or edge. The Delete Objective menu item describe later can also be used to delete an area objective, but requires selecting the area objective directly. The Delete Area menu item was included because often the space itself obscures area objective graphics, making them

difficult to select.

Delete Proportion. The Delete Proportion menu item is used to delete a proportion objective from a space, similar to the delete area item described above.

Delete All Alignment Objectives. The Delete All Alignment Objectives menu item is used to delete all alignment objectives in the current plan. This item is useful because alignment objectives are geometric objectives applied to multiple spaces, and if the topological relationship between those spaces changes the alignment objective may be obsolete. Instead of requiring them to be deleted one at a time, this item allows them to be deleted all at once.

Reset All Alignment Objectives. The Reset All Alignment Objectives menu item resets all integral spring forces to zero. As described in section 4.2.7.1, an integral spring continually adds or subtracts a force until a desired condition is met. This button can be used when a dynamical simulation becomes unstable due to a chaotic event, such as when two spaces ‘stuck’ at a corner become ‘unstuck’.

A.2.4.3. All objectives

Delete Objective. The Delete Objective menu item is used to delete any objective created with the previously described menu items. After the item is selected, select an objective.

A.2.5. Debug menu

The Debug menu, shown in figure A-13, controls the printing of a wide variety of information about the state of the dynamic simulation. As the name implies this information was used during prototype development for debugging purposes. Debugging information can be output to either a Text Editor window (figure A-27), standard output, or to a file named output.txt. The output method is set with the

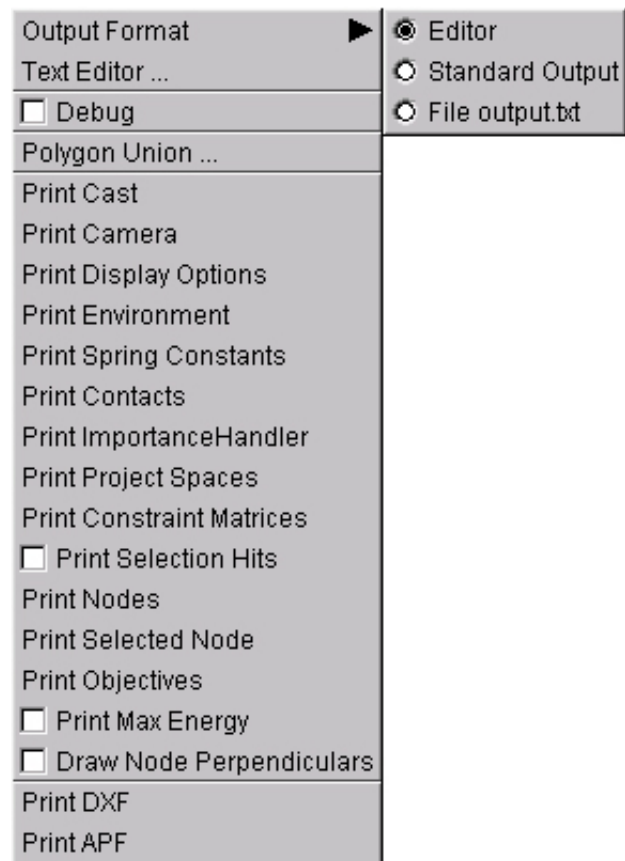


Fig. A-13. Debug menu

Output Format menu item under the Debug menu.

See also Sections A.5.4, Text Editor, and A.5.5, Debug Polygon Union, for additional dialogs available through the Debug menu.

A.2.6. Help menu

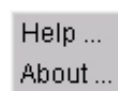


Fig. A-14. Help menu

Help. The Help menu item is used to display the Help dialog window. See section A.5.6.

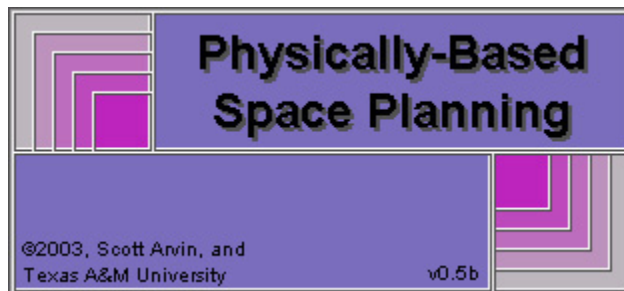


Fig. A-15. About panel

About. The About menu item is used to display the About Panel, shown in figure A-15. It displays the name of the program, copyright information, and the date of the build.

A.3. Simulation controls

The Simulation controls, shown in figure A-16, provide an interface that in general controls the dynamic simulation environment. These controls do not create or modify space elements and objectives, but control how these elements behave.

Start/Stop button. The Start button starts a dynamic simulation, which continuously advances the time by one time step and computes the next state. Once pressed its label changes to Stop. Clicking on the Stop button stops the dynamic simulation and its label changes to Start.

Step button. The Step button causes the dynamic simulation to advance the time one time step and compute a single new frame. If the dynamic simulation was

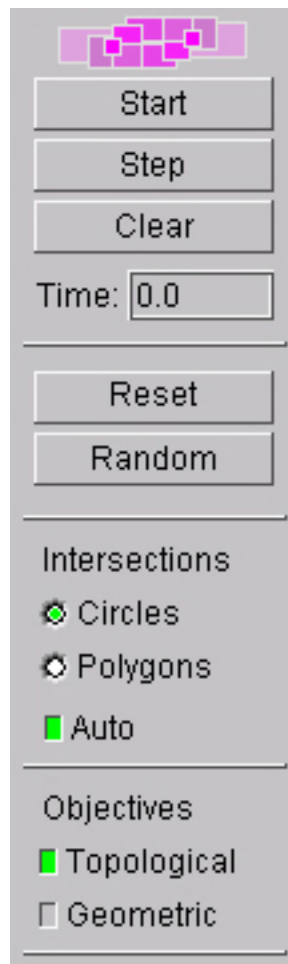


Fig. A-16. Simulation controls

already continuously running it will stop running and take one more step.

Clear button. The Clear button is used to remove all spaces and their objectives from the current project space. It is similar to the New menu item on the File menu (section A.2.1).

Time. The Time displays the current time during a dynamic simulation. The simulation timestep is $1/30$ of a second, intending to create a rate of 30 frames per second. While a simulation is running a clock is displayed in the drawing area near the time, as shown in figure A-17.



Fig. A-17. Drawing area clock

Reset button. The Reset button resets the state of the dynamic simulation to its initial conditions. The time is set to zero, and all spaces relocate to their initially specified positions.

Random button. The Random button is similar to the Reset button, except each space is set to a new random position.

Intersections radio buttons. The Intersections radio buttons set the current collision detection method used to determine when two spaces have collided with each other. The **Circles** and **Polygons** button toggles the collision detection method between the circular representation of spaces (section 4.3.1) and their actual polygonal shape (section 4.3.2). Clicking the Circles button also turns on Topological Objectives and turns off Geometric Objectives. Clicking the Polygons button does the opposite. The **Auto button** causes the simulation to switch automatically from circle intersections to polygonal intersections when the simulation has reached dynamic equilibrium, as described in section 4.3.

Objectives buttons. The Objectives buttons toggle the use of **Topological** and **Geometric** Objectives, respectively. Turning off the Topological button will disable all topological objectives. Turning off both objectives will disable all

objectives, resulting in a completely static plan.

For some space planning problems, especially those with many adjacency requirements for each room, if adjacency objectives are still active during geometric resolution they may have the effect of compressing the entire space plan. For this reason it is useful to manually turn them off by toggling the Topological button.

Coordinates. When the right mouse button is pressed in an empty area of the drawing area the current cursor coordinates are displayed at the bottom of the simulation control area, to the left of the drawing area.

A.4. Display tabs

The Display Tabs control the display of almost all elements drawn in the drawing area, as well as setting the maximum forces for each type of force applicator.

A.4.1. Display tab

The Display Tab, shown in figure A-18, contains controls to set the visibility of various element components in the dynamic simulation.

A.4.1.1. Nodes

Nodes button. The Nodes button controls the visibility of all Nodes, Space as well as Wall.

Space button. The Space button controls the visibility of each space's center point node.

Wall button. The Wall button controls the visibility of each space edge's line node.

Node Tree button. The Node Tree button controls the visibility of the node

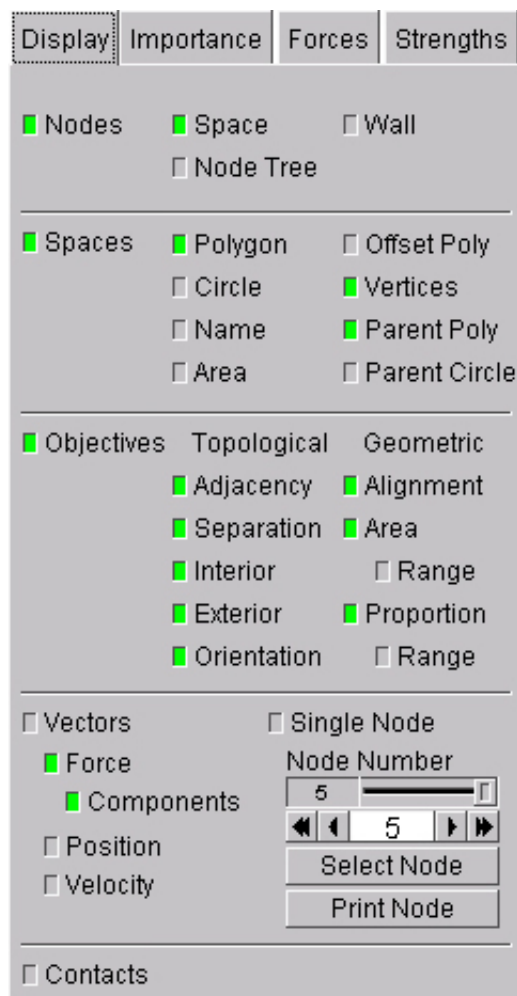


Fig. A-18. Display tab

tree, which draws lines showing the hierarchical relationship of all space and wall nodes.

A.4.1.2. Spaces

Spaces button. The Spaces button controls the visibility of all space graphic elements, not including nodes and objectives.

Polygon button. The Polygon button controls the visibility of each space's

edges.

Circle button. The Circle button controls the visibility of the circular representation of each space's area. This display is useful during topological resolution when the boundary of each space is treated as a circle (see section 4.3.1).

Name button. The Name button controls the visibility of each space's name or label.

Area button. The Area button controls the visibility of each space's area.

Offset Poly button. The Offset Poly button controls the visibility of the exterior offset of each space's edges. A space's offset polygon is used in the calculation of the building's exterior wall surface by finding the union of all such polygons. (Interior walls are formed naturally by the colliding edges of adjacent spaces.)

Vertices button. The Vertices button controls the visibility of points at the intersection of space edges.

Parent Poly button. The Parent Poly button controls the visibility of a space's exterior wall surface.

Parent Circle button. The Parent Circle button controls the visibility of a space's parent's smallest enclosing circle. This circle is used by interior, exterior, and orientation objectives to determine appropriate forces.

A.4.1.3. Objectives

Objectives button. The Objectives button controls the visibility of all objectives.

Other buttons in the Objectives section control the visibility of all their respective objectives.

Range buttons. Area and proportion objectives are drawn with dashed rectangles representing a space's target area and proportion, respectively. Each of these objectives can potentially be defined with a target range, such as $\pm 10\%$. The Range

buttons control the visibility of these target ranges.

A.4.1.4. Vectors

Vectors button. The Draw Vectors button controls the visibility of all vectors (position, velocity, and force vectors). Vectors cannot be drawn until one frame of the dynamic simulation has been computed. If the current simulation is not running, toggling Vectors on will have no immediate affect. To get the force vectors to display, click the Step button on the simulation controls to advance the simulation by one time step.

Note that the display of vectors can significantly slow down the simulation.

Force button. The Force button controls the visibility of force vectors on nodes.

Components button. The Components button controls the visibility of force components on nodes if the force button is selected. If the component button is unselected, then a single vector representing the total force on each node is displayed. If it is selected, then each individual component of that force is displayed as a separate vector.

Position button. The Position button controls the display of node position vectors, which are displayed as a line from a node's position at the end of the last time step to its current position.

Velocity button. The Velocity button controls the display of node velocity vectors, which are displayed as a line from a node's current position in the direction of the current velocity, with a length equal to the magnitude of the velocity.

Single Node button. The Single Node button toggles vectors to display for either all nodes or for only a single selected node. The selected node is specified by clicking on the **Select button** and selecting a space or wall node in the drawing

area.

A.4.1.5. *Contacts*

The **Contacts** button controls the visibility of contact points between spaces during geometric resolution.

A.4.2. *Importance tab*

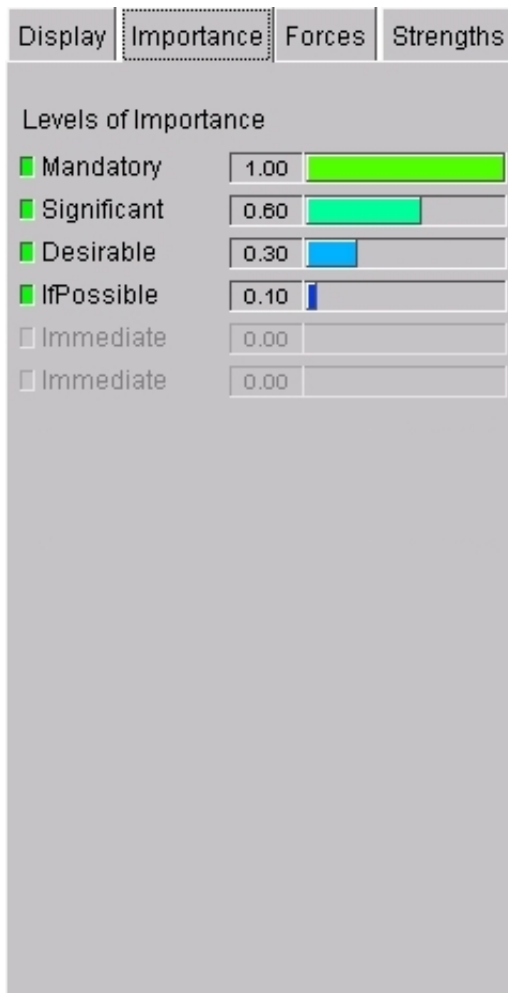


Fig. A-19. Importance tab

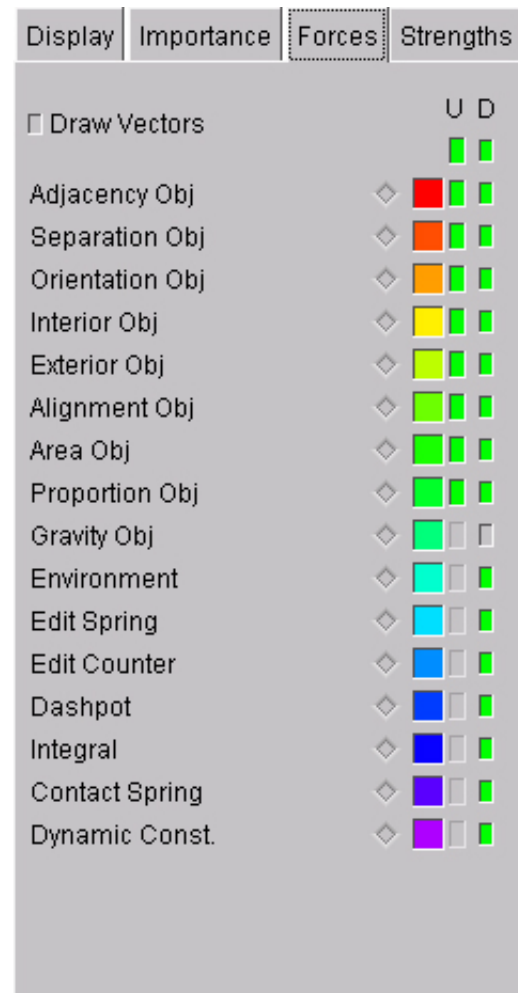


Fig. A-20. Forces tab

The Importance tab, shown in figure A-19, sets the relative strength of different objectives, as described in section 4.2.1.4. The descriptive terms for the different levels of importance are hard coded to Mandatory, Significant, Desirable, and If-Possible. The values for each are set in by manipulating the sliders on the right. The buttons to the left of the label control the visibility of all objectives with the associated level of importance.

A.4.3. Forces tab

The Forces tab, shown in figure A-20, controls the use and visibility of individual forces. Each force used in the prototype is listed on the left, and their corresponding controls are on the right. Figure A-21 shows a sample simulation state with force vectors displayed.

Draw Vectors button. The Draw Vectors button toggles the display of force vectors in the drawing area, and is a duplicate of the Force button described in section A.4.1.4.

In-use diamonds. The In-use diamonds directly to the right of each force label indicate whether any forces of this type exist in the current simulation state.

Color buttons. The Color button indicates the color used to display the associated force vector. The color can be changed by clicking on the button, which will display a Color Selector dialog as shown in figure A-22, and selecting or setting the desired color. The colors used to draw force vectors are set by default to be evenly divided around the hue part of the color wheel. This control was useful during development to highlight specific force types, since some of these colors are visually close together.

Use (U) buttons. The Use buttons, under the ‘U’ label, control whether or not to use or apply the associated forces in the dynamic simulation.

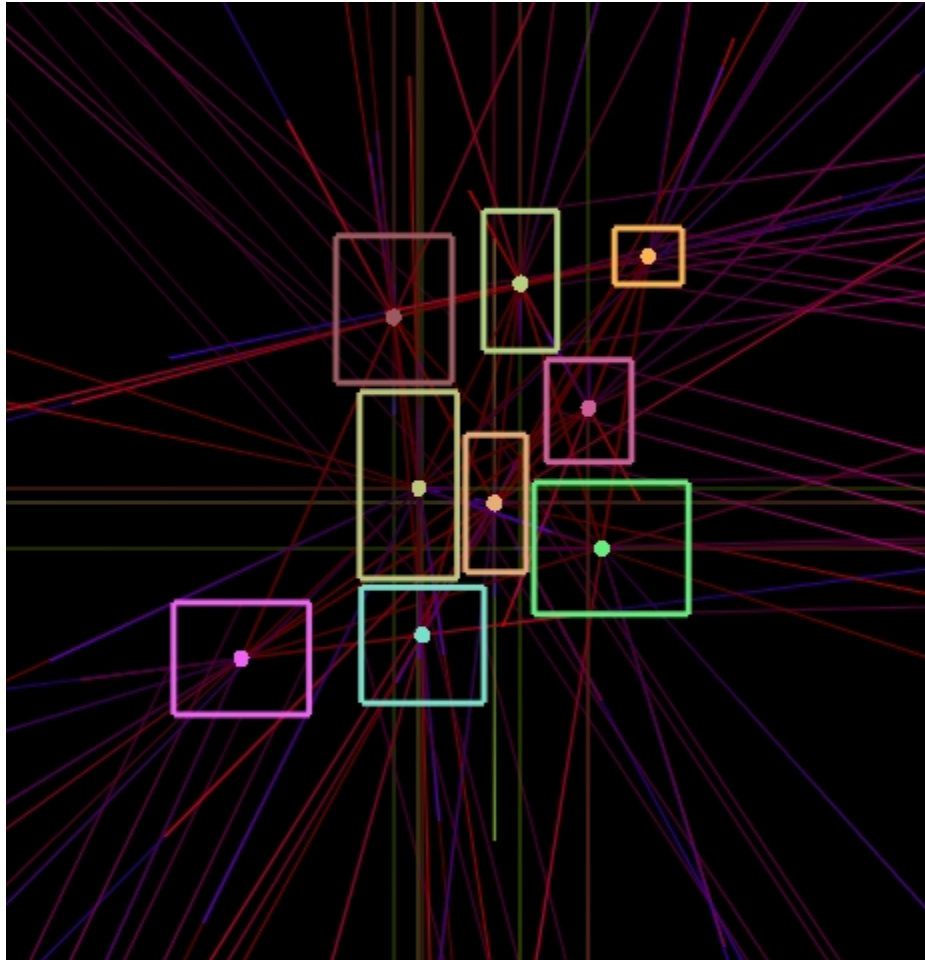


Fig. A-21. Example of a space plan showing individual force vectors

Display (D) buttons. The Display buttons, under the ‘D’ label, control whether or not to display the individual force vectors for the associated forces in the dynamic simulation.

A.4.4. *Strengths tab*

The Strengths tab, shown in figure A-23, is used to set the relative strengths of the various force types. Each force used in this prototype is listed on the left similar to the Forces tab, with a corresponding slider on their right. The available range is

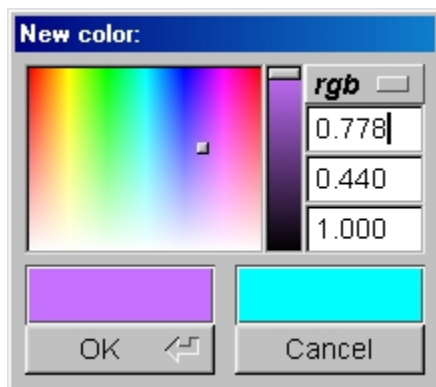


Fig. A-22. Color selector

preset for each force type and is not customizable.

A.5. Dialogs

The following sections describe the various dialog windows available for display through menu items.

A.5.1. *Font dialog*

The Font dialog, accessed from the Options menu and shown in figure A-24, is used to specify font characteristics for displaying space names.

Face. The Face selection box displays all true type fonts (*.ttf) currently in the Fonts directory within the same directory where the Physically Based Space Planning application resides.

Type. The Type radio buttons control the way the fonts are displayed. Available choices are Filled, Outline, Bitmap, and Pixmap. These choices were those available in the gltt library (GLTT, 2001) that made possible the drawing of true type fonts in OpenGL. The Bitmap and Pixmap options, however, do not work.

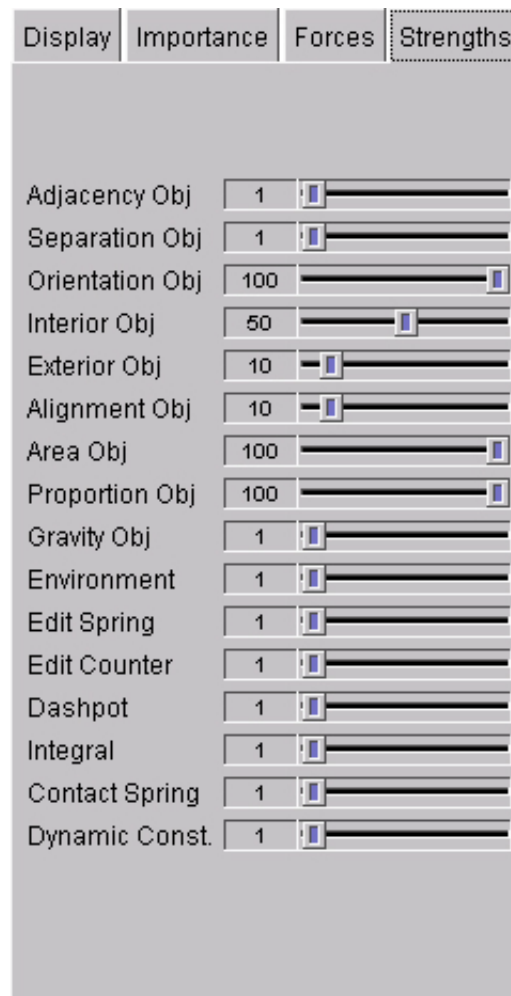


Fig. A-23. Strengths tab

Size. The Size edit box controls the size, in pixels, of displayed text.

A.5.2. *Thumbnails dialog*

The Thumbnails dialog, accessed from the Options menu and shown in figure A-25, can be used while designing a space plan to save and restore potential designs. It is intended to be used during an experimental test to determine the level of support for design emergence, and its corollary, the level of reduction of design fixation.

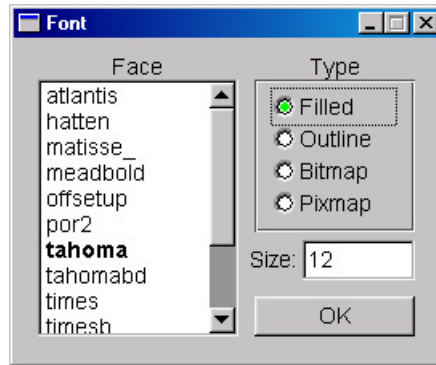


Fig. A-24. Font dialog

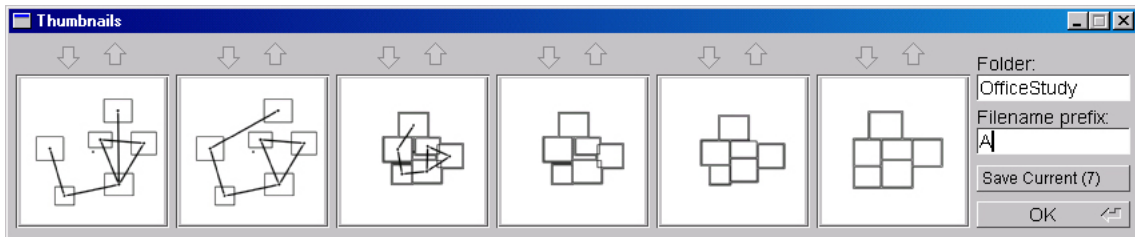


Fig. A-25. Thumbnails dialog

Six different numbered plans can be saved and restored using the down and up arrows, respectively. The **Folder** edit box is used to specify the location where the different plans and images will be stored. The **Filename Prefix** edit box is used to specify the prefix for each file created. For example, if the first down arrow is selected, the space plan currently displayed in the drawing area will be saved as “<Folder>/<Filename Prefix> - 1.apf”, and an image of the current drawing area will be saved as “<Folder>/<Filename Prefix> - 1.png”. A running index is kept of the number of files saved, and additional files are saved in history folders so that a history of the design process can be viewed. If the current index is 1, then the files “<Folder>/HistoryApf/<Filename Prefix> - 001 - 1.apf” and “<Folder>/HistoryPng/<Filename Prefix> - 001 - 1.png” are saved along with those

above. Continuing the example, if the third down arrow is selected, the following files would be created:

<Folder>/<Filename Prefix> - 3.apf
 <Folder>/<Filename Prefix> - 3.png
 <Folder>/HistoryApf/<Filename Prefix> - 002 - 3.apf
 <Folder>/HistoryPng/<Filename Prefix> - 002 - 3.png

If thumbnail 1 is saved again, the following files would be created:

<Folder>/<Filename Prefix> - 1.apf
 <Folder>/<Filename Prefix> - 1.png
 <Folder>/HistoryApf/<Filename Prefix> - 003 - 1.apf
 <Folder>/HistoryPng/<Filename Prefix> - 003 - 1.png

Note that in the last save the first two files replace those created during the first save, but that the history files are new. In this way the six most promising plans are saved and can be reviewed, as well as the entire history of their creation.

The **Save Current** button is used to save the current space plan without using one of the thumbnails, and uses the current index described above. Continuing the example from above, if the Save Current button was selected with a current index of 4, the following files would be created:

<Folder>/HistoryApf/<Filename Prefix> - 004 - Save.apf
 <Folder>/HistoryPng/<Filename Prefix> - 004 - Save.png

This button was intended to be used by an observer during an experiment to

save an interesting plan that the user would not have saved.

A.5.3. Settings dialog

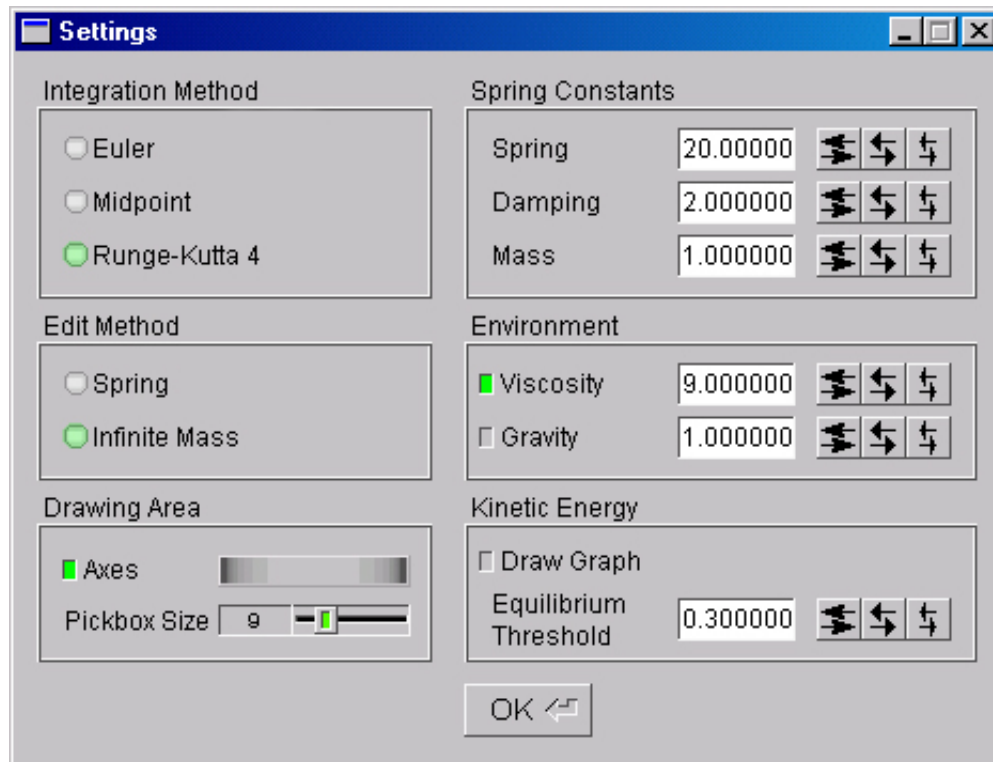


Fig. A-26. Settings dialog

The Settings dialog, accessed from the Options menu and shown in figure A-26, controls various aspects of the simulation environment described in the following sections.

A.5.3.1. Integration Method

The Integration Method radio buttons sets the numerical integration method used in the dynamic simulation, as described in section 3.3.5.2. The better the quality of the numerical integration, the more stable the dynamic system. The three options available are **Euler**, **Midpoint**, and **Runge-Kutta 4**, each successive option having a higher quality.

A.5.3.2. Edit Method

The Edit Method setting controls the method used to provide user interaction with mass elements (nodes), as described in section 3.3.8. The two options are Spring and Infinite Mass.

Spring. If the Spring option is selected, when the user clicks and drags a space's center node, a spring with zero length is temporarily created connecting the cursor location with the space's center node. As the cursor moves the spring exerts a force on the space node to move it in the direction of the cursor. The spring is removed when the user releases the mouse.

Infinite Mass. If the Infinite Mass option is selected, when the user clicks and drags a space the mass of it's center node is temporarily essentially set to infinity (in reality the mass inverse is temporarily set to zero, as explained in section 3.3.8). The user can move the space anywhere on the screen, and since its mass is infinite any forces acting upon it have no effect.

A.5.3.3. Drawing Area

The Drawing Area settings control miscellaneous settings related to the drawing area that do not fit neatly elsewhere.

Axes button and roller. The Axes button toggles the display of the X (a red line), Y (a green line), and Z (a blue line) axes ($XYZ = RGB$). Clicking on the roller and dragging to the left and right respectively decreases and increases the length of the axes lines.

Pickbox Size slider. The Pickbox slide sets the size of the pick box used to determine what elements are under the cursor when a mouse button is clicked.

A.5.3.4. Spring Constants

The Spring Constants settings control the overall spring constant, damping constant, and mass constant for the dynamic simulation.

Spring. The Spring setting controls the strength of springs, which determines the force a spring will apply to a node when the spring's current length does not equal its target or rest length. Increasing the spring constant generally increases the strength of all springs, thereby increasing the chance that an objective that uses springs will be met.

Damping. The Damping setting controls the damping strength of springs, which determines the force a spring will apply to a node when the node has a non-zero velocity. Increasing the damping constant generally decreases the time it takes for the system to come to equilibrium, because the dampers do not allow the objects to build up velocity.

Mass. The Mass setting controls the mass of all nodes. Increasing the **Mass** constant of each object also generally decreases the time it takes for the system to come to equilibrium, because it takes more force to change the velocity of the objects.

The values for all of these settings can be specified directly in the **edit boxes**, or by using the **numeric spinners** to their right. Clicking on the arrow buttons increases or decreases the associated value. Clicking increases the value, while click-

ing with the Ctrl key pressed decreases it. The left arrows adjust by 1.0, the middle arrows by 0.1, and the right arrows by 0.01.

The relationship between these parameters affects the overall stability of the dynamics system, as described in section 3.3.5.3. The effects described above only work within a limited range, beyond which the system becomes unstable and unusable.

A.5.3.5. Environment

The Environment settings control the overall environmental characteristics in which the dynamic simulations take place. Environmental characteristics generally apply to every element in a simulation, as opposed to other characteristics that apply only to selective elements. Without these environmental characteristics, no outside forces are applied to the elements in the dynamic simulation, so they are essentially moving in outer space.

The buttons control whether or not the associated value has any effect, essentially setting it to zero. The numeric spinners work as described above in section A.5.3.4.

Viscosity. The Viscosity setting simulates the presence of air around the elements, and results in a force due to viscous drag being applied to all elements. The magnitude of this force is directly proportional to an element's velocity, while the direction of the force is opposite that of the velocity. See section 3.3.3.1 for a more complete description.

Without viscosity and without any other forces acting on it, then according to Newton's first law of motion, once an element has a velocity it continues in motion until another force is applied to it. This kind of behavior is unusual in our experience, and makes dynamical simulations less 'believable.' Making the environment generally viscous improves the believability of the simulation, and makes it easier to interact

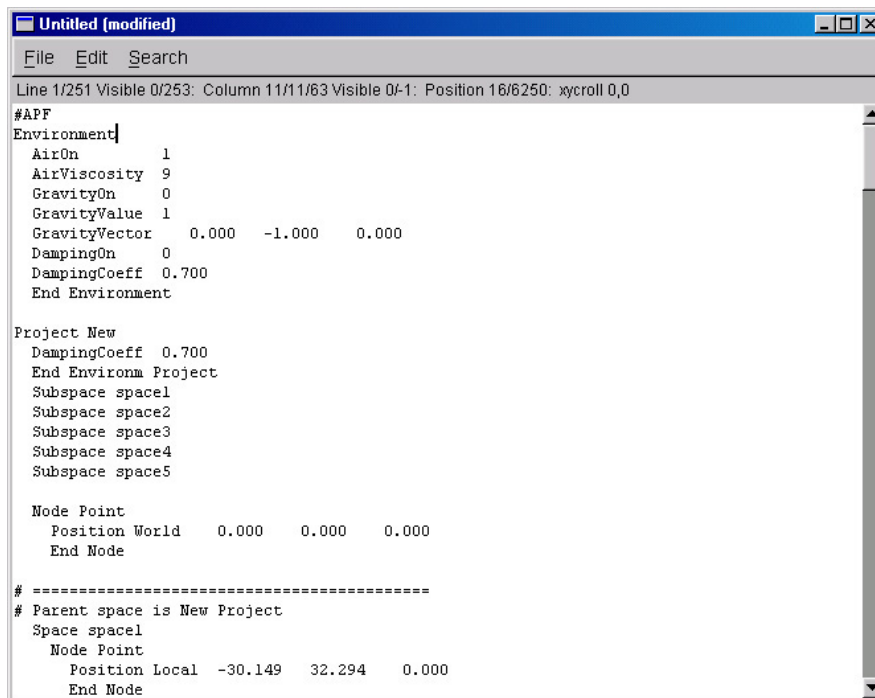
with the elements because they tend to reach equilibrium faster.

Gravity. The Gravity setting simulates the presence of a massive body near the elements, and results in a constant force being applied to all elements. To simulate a system near Earth, the magnitude of the force is constant, and the direction is constantly downward. Using gravity is important in some dynamics simulations where the behavior of falling bodies is being simulated. This particular behavior is not important in physically based space planning.

A.5.3.6. *Kinetic Energy*

Draw Graph button. The Draw Graph button toggles the display of the kinetic energy graph in the drawing window.

Equilibrium Threshold. The Equilibrium Threshold setting defines when a



```

Untitled (modified)
File Edit Search
Line 1/251 Visible 0/253: Column 11/11/63 Visible 0/1: Position 16/6250: xycroll 0,0
#APF
Environment
  AirOn 1
  AirViscosity 9
  GravityOn 0
  GravityValue 1
  GravityVector 0.000 -1.000 0.000
  DampingOn 0
  DampingCoeff 0.700
End Environment

Project New
  DampingCoeff 0.700
End Environm Project
Subspace space1
Subspace space2
Subspace space3
Subspace space4
Subspace space5

Node Point
  Position World 0.000 0.000 0.000
End Node

# =====
# Parent space is New Project
Space space1
  Node Point
    Position Local -30.149 32.294 0.000
  End Node

```

Fig. A-27. Text editor window

simulation achieves dynamic equilibrium, as described in section 4.3. The numeric spinners work as described above in section A.5.3.4.

A.5.4. *Text editor window*

The text editor, accessed from the Debug menu and shown in figure A-27, is used to display a variety of debug information. If the current debug output format is set to text editor, any output from the debug commands will be displayed in the this window. It will not be further described.

A.5.5. *Debug Polygon Union dialog*

The Debug Polygon Union dialog, accessed from the Debug menu and shown in figure A-28, was used during the development of the Polygon Union algorithm described in appendix E. This algorithm is used to create the parent polygon that displays an outline around a set of spaces, in effect displaying the exterior wall surfaces of a building (see section A.1.1). It will not be further described.

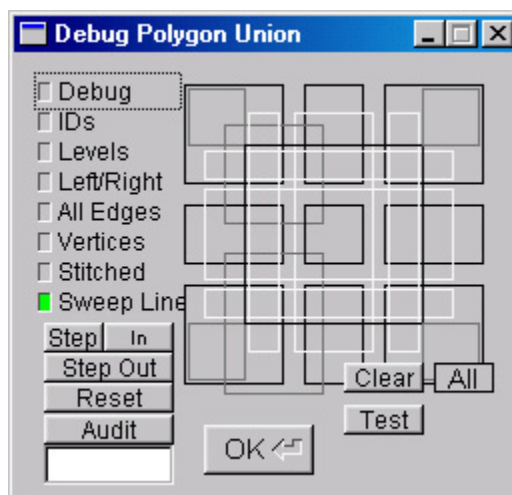


Fig. A-28. Debug Polygon Union dialog

A.5.6. Help dialog

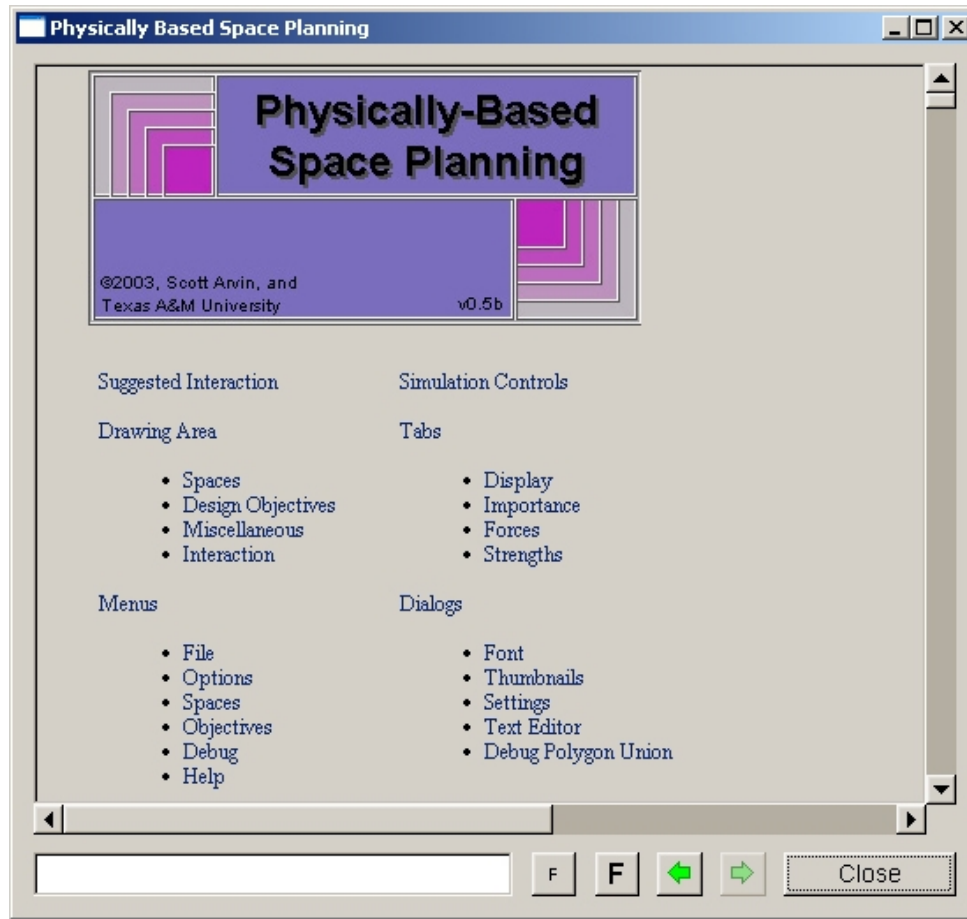


Fig. A-29. Help dialog

The Help dialog, accessed from the Help menu and shown in figure A-29, is used to display help text in HTML format.

APPENDIX B

APF FILE FORMAT

An .apf (Architectural Programming File) file contains programming information about an architectural design project. It is intended as a sample input file for a Physically Based Space Planning system. See figure B-30 for a very simple example. See also Appendices C and D for more complicated examples and sections 5.3 and 5.4 for some samples of their use.

```
#APF Version 0.0 Project project name
  Subspace space 1
  Subspace space 2

Space space 1
  Adjacency Immediate space 2
  Subspace space 3

Space space 2
  Adjacency Immediate space 1

Space space 3
```

Fig. B-30. A simple .apf file

The first line identifies the file as an apf formatted file. The rest of the file contains a definition of each ‘Space’ in the design project in the form of a tree. A space can be anything from an entire site, all the way down to a piece of paper on a desk. Any space may contain any number of subspaces that are wholly contained within it, each of which can contain their own subspaces. In this way, a hierarchical structure of parent-child spatial relationships can be constructed. Each space typically defines a number of design objectives for it, such as area requirements and adjacency rela-

tionships. The top level or root space is the project space, which should not contain any relationships to other spaces (except for a list of child spaces, of course). Except for the project space, all spaces must first be referred to in a Subspace_Name structure, and later as a complete definition in a space. In this way, the file represents a top down definition of the tree structure.

Note that in the prototype described here only one hierarchical level is implemented; that is, subspaces are only referred to by the project space.

This appendix describes in detail the elements necessary to construct a valid apf file. The elements of an apf file are *structures*, which contain one or more *substructures*, which contain one or more *primitives*.

B.1. Symbols used

The following symbols are used to define structures, substructures, and primitives.

B.1.1. << double_angle_bracket >>

Indicates a subordinate structure is to be substituted in place of the enclosing double angle brackets. The substitute structure is found in section B.3: Structures or in alphabetical order in section B.4: Substructures.

For example, in the following definition of the APF File structure, double angle brackets are used to indicate that ‘Header’ is a structure or substructure.

```
<<Header>> {1:1}
<<Project>> {1:1}
<<Space>> {1:m}
```

B.1.2. < single_angle_bracket >

Indicates the name of a value for this line: < Primitive >. The specific definition of this value is found in alphabetical order in section B.5: Primitive Elements. For

example, in the following definition of a Polygon, single angle brackets are used to indicate that ‘Integer’ and ‘Point’ primitives are required.

```
Polygon <Integer>
      <Point> {3:m}
```

B.1.3. { braces }

Indicates the minimum to maximum occurrences allowed for this structure or line: Minimum:Maximum . Note that minimum and maximum occurrence limits are defined relative to the enclosing structure. This means that a required line (minimum = 1) is not required if the optional enclosing structure is not present. Similarly, a line occurring only once (maximum = 1) may occur multiple times as long as each occurs only once under its own multiple-occurring structure. An ‘m’ indicates that ‘many’ occurrences are allowed. For example, in the previous definition of a Polygon, braces indicate that 3 or more Point elements are required.

If no occurrence indication is noted next to an item, one item is required; that is, assume 1:1.

B.1.4. [square brackets]

Indicates a choice of one or more options: [Choice of]. A ‘-’ separating two values indicates a range of values, for example [0-9].

B.1.5. | vertical bar

Separates multiple choices, for example [Choice 1 | Choice 2].

B.2. General

All lines within structures and substructures should end with a new line character.

B.2.1. Comment

A line is considered a comment if it begins with '#'. Comments and blank lines may be placed on any line in the file, except the first line, which identifies the file type.

```
# <String> {0:1}
```

B.3. Structures

B.3.1. APF File

```
#APF <Version_Number>
<<Environment>> {1:1}
<<Energy_Graph>> {1:1}
<<Project>> {1:1}
#End APF
```

B.3.2. Project

```
Project <Space_Name>
<<Subspace_Name>> {1:m}
<<PointNode>> {1:1} Must come before Polygon
<<Polygon>> {0:1} Default: Randomly Positioned Rectangle
<<Area_Objective>> {0:1}
<<Proportion_Objective>> {0:1} Default: 1.0
<<Space>> {0:m}
```

Used in APF File.

B.3.3. Space

```
Space <Space_Name>
<<Subspace_Name>> {0:m}
<<PointNode>> {1:1} Must come before Polygon
<<Polygon>> {0:1} Default: Randomly Positioned Rectangle
[<<Interior_Objective>> | <<Boundary_Location_Objective>>] {0:1}
<<Area_Objective>> {0:1}
<<Proportion_Objective>> {0:1} Default: 1.0
<<Adjacency_Objective>> {0:m}
<<Separation_Objective>> {0:m}
<<Alignment_Objective>> {0:m}
<<Space>> {0:m}
```

Used in Project and Space.

B.4. Substructures

B.4.1. *Adjacency_Objective*

Adjacency <Importance> <Space_Name>

Used in Space.

B.4.2. *Alignment_Objective*

[Alignment | Offset <Float>] <Importance> (cont.)
<Integer> <Integer> <Space_Name>

Used in Space.

B.4.3. *Area_Objective*

Area <Float> [<Tolerance>]

Used in Project, and Space.

B.4.4. *Boundary_Location_Objective*

[<<Exterior_Objective>> | <<Orientation_Objective>>]

Used in Space.

B.4.5. *Energy_Graph*

```
EnergyGraph
  Position <Vector>
  Width <Float>
  Height <Float>
  drawGraph [ 0 | 1 ]
End EnergyGraph
```

Used in APF File.

B.4.6. Environment

```

Environment
  AirOn [ 0 | 1 ]
  AirViscosity <Float>
  GravityOn [ 0 | 1 ]
  GravityValue <Float>
  GravityVector <Vector>
End Environment

```

Used in APF File.

B.4.7. Exterior_Objective

```

Exterior <Importance>

```

Used in Boundary_Location_Objective.

B.4.8. Interior_Objective

```

Interior <Importance>

```

Used in Space.

B.4.9. Line_Node

```

Node Line
<<Node>>
Direction <<Vector>>
End Node

```

Used in Polygon.

B.4.10. Node

```

Position <Position> {1:1} Default: Local
<Mass> {1:1} Default: 1.0
Nailed {0:1} Default: Unnailed

```

Used in Line_Node, and Point_Node.

B.4.11. Orientation_Objective

Orientation <Importance> <Direction>

Used in Boundary_Location_Objective.

B.4.12. Point_Node

Node Point
 <<Node>>
 End Node

Used in Project, and Space.

B.4.13. Polygon

Polygon <Integer>
 <Point_Type> <Point> {3:m}
 <<LineNode>> {Number of points}

The Integer indicates the number of vertices for this polygon.

Used in Project, Space.

B.4.14. Proportion_Objective

Proportion <Float> [<Tolerance>]

The proportion is the ratio of the maximum width of the polygon divided by the maximum height.

Used in Project, Space.

B.4.15. Separation_Objective

Separation <Importance> <Space_Name>

Used in Space.

B.4.16. Subspace_Name

Subspace <Space_Name>

A space that is enclosed within the boundary of a parent space. A space can only be a subspace within at most one parent space, and the Space for a space should be specified *after* the Subspace_Name that refers to it.

Used in Project, and Space.

B.5. Primitives*B.5.1. Angle*

<Float>

Value will be converted to the range [0.0, 360.0).

Used in Direction.

B.5.2. Compass_Direction

[N | S | E | W | NE | NW | SE | SW]

Used in Direction.

B.5.3. Digit

[0-9]

Used in Number.

B.5.4. Direction

[<Compass_Direction> | <Angle>]

Used in Orientation_Objective.

B.5.5. Float

`<Number>.<Number>`

Used in Proportion_Objective, Environment, Importance, Angle, Mass, Alignment_Objective, Area_Objective, Vector, and Point.

B.5.6. Importance

`[[Mandatory | Significant | Desirable | IfPossible] | (cont.)
Custom <Float>]`

Used in Adjacency_Objective, Orientation_Objective, Alignment_Objective, Exterior_Objective, Interior_Objective, and Separation_Objective.

B.5.7. Integer

`<Number>`

Used in Area_Objective, Alignment_Objective, Polygon, Tolerance.

B.5.8. Mass

`Mass [Infinite | <Float>]`

Used in Node.

B.5.9. Number

`[<Digit> | <Number><Digit>]`

Used in Integer, Float.

B.5.10. Point

`<Float> <Float> <Float>`

A three dimensional coordinate in the world coordinate system.

Used in Polygon.

B.5.11. Point_Type

[X | x | T | t | P | p]

Indicates the type of point for a polygon.

X or x: the intersection of two Line Nodes, which will be created using the surrounding vertices.

T or t: a transition point between a Line Node and a Point Node. A Point Node will be created at this point.

P or p: a Point Node, which will be created at this point.

(Note: Transition and Point nodes for polygons are not currently implemented.)

Used in Polygon.

B.5.12. Position

[Local | World]

Used in Node.

B.5.13. Space_Name

<String>

A string identifying the name of a space.

Used in Project, Space, Subspace_Name, Alignment_Objective, Separation_Objective, and Adjacency_Objective.

B.5.14. String

[any number of characters that are not a newline character]

Used in Space_Name.

B.5.15. Tolerance

<Integer>

The tolerance of a previously specified value, as a percent. For example, an area objective of 120 square feet might have a tolerance of +/- 10

(Not implemented.)

Used in Area_Objective, Proportion_Objective.

B.5.16. Version_Number

Version [0.0]

Used in APF File.

B.5.17. Vector

<Float> <Float> <Float>

Used in Polygon, Line_Node, and Environment.

APPENDIX C

APF SAMPLE FILE - COUNSELING CENTER

#APF Version 0.0
 # From Mark Karlen, Space Planning Basics, p. 22

Project University Career Counseling Center

- Subspace Reception
- Subspace Interview Station
- Subspace Director
- Subspace Staff
- Subspace Seminar Room
- Subspace Rest Room
- Subspace Work Area
- Subspace Coffee Station
- Subspace Guest Apartment

Space Reception

- Area 250
- Aspect 1.35
- Adjacency Immediate Interview Station
- Adjacency Important Director
- Adjacency Convenient Staff
- Adjacency Immediate Seminar Room
- Adjacency Convenient Rest Room
- Adjacency Convenient Coffee Station
- Separation Immediate Guest Apartment
- Exterior

Space Interview Station

- Area 220
- Aspect 0.65
- Adjacency Immediate Reception
- Adjacency Convenient Director
- Adjacency Immediate Staff
- Adjacency Convenient Rest Room
- Adjacency Convenient Coffee Station
- Separation Immediate Guest Apartment
- Exterior

Space Director

- Area 140
- Aspect 0.7
- Adjacency Important Reception
- Adjacency Convenient Interview Station
- Adjacency Immediate Staff
- Adjacency Convenient Seminar Room
- Adjacency Convenient Work Area
- Adjacency Convenient Coffee Station
- Separation Immediate Guest Apartment
- Exterior
- Orientation NE

Space Staff

- Area 180
- Aspect 0.7
- Adjacency Convenient Reception
- Adjacency Immediate Interview Station
- Adjacency Immediate Director

Adjacency Convenient Rest Room
 Adjacency Important Work Area
 Adjacency Convenient Coffee Station
 Separation Immediate Guest Apartment
 Exterior

Space Seminar Room

Area 300
 Aspect 1.2
 Adjacency Immediate Reception
 Adjacency Convenient Director
 Adjacency Important Rest Room
 Adjacency Convenient Coffee Station
 Separation Immediate Guest Apartment
 Exterior

Space Rest Room

Area 200
 Aspect 0.8
 Adjacency Convenient Reception
 Adjacency Convenient Interview Station
 Adjacency Convenient Staff
 Adjacency Important Seminar Room
 Adjacency Convenient Work Area
 Separation Immediate Guest Apartment

Space Work Area

Area 120
 Aspect 1.1
 Adjacency Convenient Director
 Adjacency Important Staff
 Adjacency Convenient Rest Room
 Adjacency Convenient Coffee Station
 Separation Immediate Guest Apartment

Space Coffee Station

Area 50
 Aspect 0.9
 Adjacency Convenient Reception
 Adjacency Convenient Interview Station
 Adjacency Convenient Director
 Adjacency Convenient Staff
 Adjacency Convenient Seminar Room
 Adjacency Convenient Work Area
 Separation Immediate Guest Apartment
 Exterior

Space Guest Apartment

Area 350
 Aspect 1.0
 # test comment
 Separation Immediate Reception
 Separation Immediate Interview Station
 Separation Immediate Director
 Separation Immediate Staff
 Separation Immediate Seminar Room
 Separation Immediate Rest Room
 Separation Immediate Work Area
 Separation Immediate Coffee Station
 Exterior

APPENDIX D

APF SAMPLE FILE - RESIDENCE

```

#APF
Environment
  AirOn      1
  AirViscosity 9
  GravityOn  0
  GravityValue 1
  GravityVector 0.000  -1.000  0.000
End Environment

EnergyGraph
  Position -40.000  5.000  0.000
  Width 90.000
  Height 40.000
  drawKineticEnergy      1
  drawPotentialEnergy    0
  drawAverageEnergy      0
  drawAverageAverageEnergy 0
  drawMaxEnergy          0
  drawSeparateGraphs    0
  drawGraph              0
End EnergyGraph

Project Worked Example
  Subspace Entrance
  Subspace Living
  Subspace Family
  Subspace Dining
  Subspace Garage
  Subspace Kitchen
  Subspace Breakfast
  Subspace Laundry
  Subspace Study
  Subspace Master Bed
  Subspace Guest Bath
  Subspace Master Closet
  Subspace Master Bath
  Subspace Bedroom 1
  Subspace Bedroom 2
  Subspace Guestroom
  Subspace Bath
  Subspace Hall

Node Point
  Position World 0.000  0.000  0.000
End Node

# =====
# Parent space is Worked Example
Space Entrance
Node Point
  Position Local 2.397  -44.530  0.000
End Node

Polygon 4
  X 6.493  -41.478  0.000
  X -1.699  -41.478  0.000

```

X -1.699 -47.582 0.000
 X 6.493 -47.582 0.000

Node Line
 Position Local 4.096 0.000 0.000
 Direction 0.000 1.000 0.000
 End Node

Node Line
 Position Local 0.000 3.052 0.000
 Direction -1.000 0.000 0.000
 End Node

Node Line
 Position Local -4.096 0.000 0.000
 Direction 0.000 -1.000 0.000
 End Node

Node Line
 Position Local 0.000 -3.052 0.000
 Direction 1.000 0.000 0.000
 End Node

Area 50.0000 0.2000

Proportion 1.3421 0.2000

Adjacency Mandatory Living

=====

Parent space is Worked Example

Space Living

Node Point
 Position Local 5.550 18.291 0.000
 End Node

Polygon 4

X 16.209 25.328 0.000
 X -5.108 25.328 0.000
 X -5.108 11.255 0.000
 X 16.209 11.255 0.000

Node Line
 Position Local 10.659 0.000 0.000
 Direction 0.000 1.000 0.000
 End Node

Node Line
 Position Local 0.000 7.036 0.000
 Direction -1.000 0.000 0.000
 End Node

Node Line
 Position Local -10.659 0.000 0.000
 Direction 0.000 -1.000 0.000
 End Node

Node Line
 Position Local 0.000 -7.036 0.000
 Direction 1.000 0.000 0.000
 End Node

Area 300.0000 0.2000

Proportion 1.5148 0.2000

Adjacency Mandatory Dining

Adjacency Mandatory Entrance

```
# =====
# Parent space is Worked Example
Space Family
Node Point
  Position Local  -20.057  36.583  0.000
  End Node

Polygon 4
X  -11.540  41.866  0.000
X  -28.575  41.866  0.000
X  -28.575  31.299  0.000
X  -11.540  31.299  0.000

Node Line
  Position Local   8.518  0.000  0.000
  Direction      0.000  1.000  0.000
  End Node

Node Line
  Position Local   0.000  5.283  0.000
  Direction     -1.000  0.000  0.000
  End Node

Node Line
  Position Local  -8.518  0.000  0.000
  Direction      0.000 -1.000  0.000
  End Node

Node Line
  Position Local   0.000 -5.283  0.000
  Direction      1.000  0.000  0.000
  End Node
```

Area 180.0000 0.2000

Proportion 1.6122 0.2000

Adjacency Mandatory Kitchen

```
# =====
# Parent space is Worked Example
Space Dining
Node Point
  Position Local  -18.922   9.335  0.000
  End Node

Polygon 4
X  -10.870  16.321  0.000
X  -26.974  16.321  0.000
X  -26.974   2.349  0.000
X  -10.870   2.349  0.000

Node Line
  Position Local   8.052  0.000  0.000
  Direction      0.000  1.000  0.000
  End Node

Node Line
```

```

    Position Local    0.000    6.986    0.000
    Direction    -1.000    0.000    0.000
    End Node

Node Line
    Position Local   -8.052    0.000    0.000
    Direction     0.000   -1.000    0.000
    End Node

Node Line
    Position Local    0.000   -6.986    0.000
    Direction     1.000    0.000    0.000
    End Node

Area 225.0000 0.2000

Proportion 1.1525 0.2000

Adjacency Mandatory Living

# =====
# Parent space is Worked Example
Space Garage
Node Point
    Position Local  -36.330  -43.899    0.000
    End Node

Polygon 4
X  -25.944  -34.271    0.000
X  -46.716  -34.271    0.000
X  -46.716  -53.528    0.000
X  -25.944  -53.528    0.000

Node Line
    Position Local   10.386    0.000    0.000
    Direction     0.000    1.000    0.000
    End Node

Node Line
    Position Local    0.000    9.628    0.000
    Direction     -1.000    0.000    0.000
    End Node

Node Line
    Position Local  -10.386    0.000    0.000
    Direction     0.000   -1.000    0.000
    End Node

Node Line
    Position Local    0.000   -9.628    0.000
    Direction     1.000    0.000    0.000
    End Node

Area 400.0000 0.2000

Proportion 1.0787 0.2000

Adjacency Mandatory Kitchen

# =====
# Parent space is Worked Example
Space Kitchen
Node Point
    Position Local  -39.358   14.128    0.000

```

```

End Node

Polygon 4
X -32.201 19.368 0.000
X -46.515 19.368 0.000
X -46.515 8.889 0.000
X -32.201 8.889 0.000

Node Line
Position Local 7.157 0.000 0.000
Direction 0.000 1.000 0.000
End Node

Node Line
Position Local 0.000 5.240 0.000
Direction -1.000 0.000 0.000
End Node

Node Line
Position Local -7.157 0.000 0.000
Direction 0.000 -1.000 0.000
End Node

Node Line
Position Local 0.000 -5.240 0.000
Direction 1.000 0.000 0.000
End Node

Area 150.0000 0.2000

Proportion 1.3659 0.2000

Adjacency Mandatory Garage
Adjacency Mandatory Laundry
Adjacency Mandatory Breakfast
Adjacency Mandatory Family

# =====
# Parent space is Worked Example
Space Breakfast
Node Point
Position Local -53.635 38.163 0.000
End Node

Polygon 4
X -48.822 41.539 0.000
X -58.448 41.539 0.000
X -58.448 34.786 0.000
X -48.822 34.786 0.000

Node Line
Position Local 4.813 0.000 0.000
Direction 0.000 1.000 0.000
End Node

Node Line
Position Local 0.000 3.376 0.000
Direction -1.000 0.000 0.000
End Node

Node Line

```

```

      Position Local  -4.813    0.000    0.000
      Direction      0.000   -1.000    0.000
      End Node

      Node Line
      Position Local   0.000   -3.376    0.000
      Direction       1.000    0.000    0.000
      End Node

      Area 65.0000 0.2000

      Proportion 1.4255 0.2000

      Adjacency  Mandatory Kitchen

# =====
# Parent space is Worked Example
Space Laundry
Node Point
  Position Local  -54.988   -3.349    0.000
  End Node

Polygon 4
X  -50.906   -0.287    0.000
X  -59.071   -0.287    0.000
X  -59.071   -6.411    0.000
X  -50.906   -6.411    0.000

Node Line
  Position Local   4.082    0.000    0.000
  Direction       0.000    1.000    0.000
  End Node

Node Line
  Position Local   0.000    3.062    0.000
  Direction      -1.000    0.000    0.000
  End Node

Node Line
  Position Local  -4.082    0.000    0.000
  Direction       0.000   -1.000    0.000
  End Node

Node Line
  Position Local   0.000   -3.062    0.000
  Direction       1.000    0.000    0.000
  End Node

      Area 50.0000 0.2000

      Proportion 1.3333 0.2000

      Adjacency  Mandatory Kitchen

# =====
# Parent space is Worked Example
Space Study
Node Point
  Position Local   45.665   27.122    0.000
  End Node

Polygon 4
X   51.273   33.808    0.000
X   40.057   33.808    0.000

```

X 40.057 20.435 0.000
 X 51.273 20.435 0.000

Node Line
 Position Local 5.608 0.000 0.000
 Direction 0.000 1.000 0.000
 End Node

Node Line
 Position Local 0.000 6.687 0.000
 Direction -1.000 0.000 0.000
 End Node

Node Line
 Position Local -5.608 0.000 0.000
 Direction 0.000 -1.000 0.000
 End Node

Node Line
 Position Local 0.000 -6.687 0.000
 Direction 1.000 0.000 0.000
 End Node

Area 150.0000 0.2000

Proportion 0.8387 0.2000

=====

Parent space is Worked Example
 Space Master Bed

Node Point
 Position Local 44.404 1.388 0.000
 End Node

Polygon 4
 X 50.899 10.048 0.000
 X 37.908 10.048 0.000
 X 37.908 -7.273 0.000
 X 50.899 -7.273 0.000

Node Line
 Position Local 6.495 0.000 0.000
 Direction 0.000 1.000 0.000
 End Node

Node Line
 Position Local 0.000 8.660 0.000
 Direction -1.000 0.000 0.000
 End Node

Node Line
 Position Local -6.495 0.000 0.000
 Direction 0.000 -1.000 0.000
 End Node

Node Line
 Position Local 0.000 -8.660 0.000
 Direction 1.000 0.000 0.000
 End Node

Area 225.0000 0.2000

Proportion 0.7500 0.2000

Adjacency Mandatory Master Bath

Adjacency Mandatory Master Closet

```
# =====
# Parent space is Worked Example
Space Guest Bath
Node Point
  Position Local   94.228  -20.716   0.000
  End Node

Polygon 4
X   98.700  -18.480   0.000
X   89.756  -18.480   0.000
X   89.756  -22.952   0.000
X   98.700  -22.952   0.000

Node Line
  Position Local   4.472   0.000   0.000
  Direction        0.000   1.000   0.000
  End Node

Node Line
  Position Local   0.000   2.236   0.000
  Direction       -1.000   0.000   0.000
  End Node

Node Line
  Position Local  -4.472   0.000   0.000
  Direction        0.000  -1.000   0.000
  End Node

Node Line
  Position Local   0.000  -2.236   0.000
  Direction        1.000   0.000   0.000
  End Node

Area 40.0000 0.2000

Proportion 2.0000 0.2000

Adjacency Mandatory Guestroom

# =====
# Parent space is Worked Example
Space Master Closet
Node Point
  Position Local   69.129   6.307   0.000
  End Node

Polygon 4
X   72.832   9.008   0.000
X   65.425   9.008   0.000
X   65.425   3.607   0.000
X   72.832   3.607   0.000

Node Line
  Position Local   3.703   0.000   0.000
  Direction        0.000   1.000   0.000
  End Node

Node Line
  Position Local   0.000   2.700   0.000
  Direction       -1.000   0.000   0.000
```



```

End Node

Node Line
  Position Local  -3.703  0.000  0.000
  Direction      0.000  -1.000  0.000
End Node

Node Line
  Position Local   0.000  -2.700  0.000
  Direction      1.000   0.000  0.000
End Node

Area 40.0000 0.2000

Proportion 1.3714 0.2000

Adjacency  Mandatory Master Bed

Adjacency  Mandatory Master Bath

# =====
# Parent space is Worked Example
Space Master Bath
Node Point
  Position Local  70.263  -6.055  0.000
End Node

Polygon 4
X 75.486  -2.225  0.000
X 65.041  -2.225  0.000
X 65.041  -9.885  0.000
X 75.486  -9.885  0.000

Node Line
  Position Local   5.222  0.000  0.000
  Direction      0.000   1.000  0.000
End Node

Node Line
  Position Local   0.000   3.830  0.000
  Direction     -1.000   0.000  0.000
End Node

Node Line
  Position Local  -5.222  0.000  0.000
  Direction      0.000  -1.000  0.000
End Node

Node Line
  Position Local   0.000  -3.830  0.000
  Direction      1.000   0.000  0.000
End Node

Area 80.0000 0.2000

Proportion 1.3636 0.2000

Adjacency  Mandatory Master Bed

Adjacency  Mandatory Master Closet

# =====
# Parent space is Worked Example
Space Bedroom 1

```

```

Node Point
  Position Local   50.207  -32.546   0.000
  End Node

Polygon 4
X   57.630  -27.831   0.000
X   42.783  -27.831   0.000
X   42.783  -37.261   0.000
X   57.630  -37.261   0.000

Node Line
  Position Local   7.423   0.000   0.000
  Direction      0.000   1.000   0.000
  End Node

Node Line
  Position Local   0.000   4.715   0.000
  Direction     -1.000   0.000   0.000
  End Node

Node Line
  Position Local  -7.423   0.000   0.000
  Direction      0.000  -1.000   0.000
  End Node

Node Line
  Position Local   0.000  -4.715   0.000
  Direction      1.000   0.000   0.000
  End Node

Area 140.0000 0.2000

Proportion 1.5745 0.2000

Adjacency  Mandatory Bedroom 2

Adjacency  Mandatory Bath

# =====
# Parent space is Worked Example
Space Bedroom 2
Node Point
  Position Local   72.409  -33.051   0.000
  End Node

Polygon 4
X   78.923  -28.445   0.000
X   65.894  -28.445   0.000
X   65.894  -37.656   0.000
X   78.923  -37.656   0.000

Node Line
  Position Local   6.514   0.000   0.000
  Direction      0.000   1.000   0.000
  End Node

Node Line
  Position Local   0.000   4.605   0.000
  Direction     -1.000   0.000   0.000
  End Node

Node Line
  Position Local  -6.514   0.000   0.000
  Direction      0.000  -1.000   0.000

```

```

      End Node

Node Line
  Position Local   0.000  -4.605  0.000
  Direction       1.000   0.000  0.000
  End Node

Area 120.0000 0.2000

Proportion 1.4146 0.2000

Adjacency Mandatory Bedroom 1

Adjacency Mandatory Bath

# =====
# Parent space is Worked Example
Space Guestroom
Node Point
  Position Local   96.629  -32.672  0.000
  End Node

Polygon 4
X 102.953  -26.347  0.000
X  90.304  -26.347  0.000
X  90.304  -38.997  0.000
X 102.953  -38.997  0.000

Node Line
  Position Local    6.325  0.000  0.000
  Direction         0.000  1.000  0.000
  End Node

Node Line
  Position Local    0.000  6.325  0.000
  Direction        -1.000  0.000  0.000
  End Node

Node Line
  Position Local   -6.325  0.000  0.000
  Direction         0.000 -1.000  0.000
  End Node

Node Line
  Position Local    0.000 -6.325  0.000
  Direction         1.000  0.000  0.000
  End Node

Area 160.0000 0.2000

Proportion 1.0000 0.2000

Adjacency Mandatory Guest Bath

# =====
# Parent space is Worked Example
Space Bath
Node Point
  Position Local   55.000  -21.445  0.000
  End Node

Polygon 4
X  60.334  -19.101  0.000
X  49.666  -19.101  0.000

```

X 49.666 -23.789 0.000
 X 60.334 -23.789 0.000

Node Line
 Position Local 5.334 0.000 0.000
 Direction 0.000 1.000 0.000
 End Node

Node Line
 Position Local 0.000 2.344 0.000
 Direction -1.000 0.000 0.000
 End Node

Node Line
 Position Local -5.334 0.000 0.000
 Direction 0.000 -1.000 0.000
 End Node

Node Line
 Position Local 0.000 -2.344 0.000
 Direction 1.000 0.000 0.000
 End Node

Area 50.0000 0.2000

Proportion 2.2759 0.2000

Adjacency Mandatory Bedroom 1

Adjacency Mandatory Bedroom 2

=====

Parent space is Worked Example

Space Hall

Node Point
 Position Local 105.046 3.580 0.000
 End Node

Polygon 4
 X 117.226 5.222 0.000
 X 92.866 5.222 0.000
 X 92.866 1.938 0.000
 X 117.226 1.938 0.000

Node Line
 Position Local 12.180 0.000 0.000
 Direction 0.000 1.000 0.000
 End Node

Node Line
 Position Local 0.000 1.642 0.000
 Direction -1.000 0.000 0.000
 End Node

Node Line
 Position Local -12.180 0.000 0.000
 Direction 0.000 -1.000 0.000
 End Node

Node Line
 Position Local 0.000 -1.642 0.000
 Direction 1.000 0.000 0.000
 End Node

Area 80.0000 0.2000

Proportion 7.4174 0.2000

#End APF

#####

COMMAND LOG

#####

#

```
# Mon Jun 16 16:22:24.323 2003 # ---- File Open
# Mon Jun 16 16:22:27.888 2003 # btn Name
# Mon Jun 16 16:22:44.812 2003 # ---- Add Adjacency Objective
# Mon Jun 16 16:22:44.812 2003 # ---- Add Adjacency Objective 2
# Mon Jun 16 16:22:49.920 2003 # ---- Moving Space While Not Running
# Mon Jun 16 16:22:54.656 2003 # ---- Add Adjacency Objective
# Mon Jun 16 16:22:54.656 2003 # ---- Add Adjacency Objective 2
# Mon Jun 16 16:23:02.508 2003 # ---- Moving Space While Not Running
# Mon Jun 16 16:23:10.720 2003 # ---- Add Adjacency Objective
# Mon Jun 16 16:23:10.720 2003 # ---- Add Adjacency Objective 2
# Mon Jun 16 16:23:21.375 2003 # ---- Add Adjacency Objective
# Mon Jun 16 16:23:21.375 2003 # ---- Add Adjacency Objective 2
# Mon Jun 16 16:23:29.737 2003 # ---- File Save
```

APPENDIX E

ALGORITHM TO UNION MULTIPLE POLYGONS

E.1. Problem

The problem addressed here is to find the union of an arbitrary number of polygons, and also to identify in the resulting data structure the exterior boundary and all interior islands. See figure E-31.

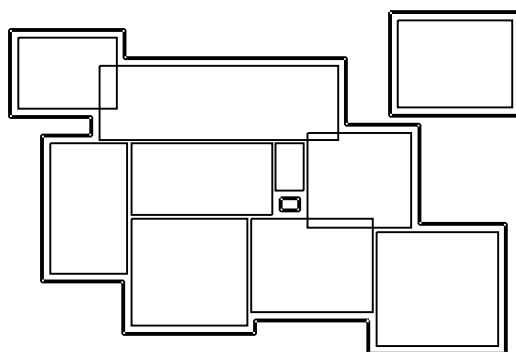


Fig. E-31. Sample problem

E.2. Previous work

The algorithms reviewed were Vatti (1992), Schutte (1995), Leonov and Nikitin (1997), and Zalik et al. (1998). Algorithms used to operate on polygons usually involve the boolean operations union, subtract, intersection, and exclusive or. All of these operations except union can only be performed on two polygons. None of the reviewed algorithms operates on an arbitrary number of polygons. Most of them

use a sweep line strategy and find the intersections between line segments, some as a preprocessing step. One combines these into a ‘scanbeam’ strategy that uses the area between two successive sweep lines to find intersections. Some algorithms iterate over ordered polygonal edges, crossing over to the other polygon at intersection points. Many use some type of edge labeling scheme. And some require geometric operations such as determining point location in a polygon.

A sweep line strategy is a method for solving geometric problems where a line is ‘swept’ through the space being considered, and operations are performed at each step of the sweep such that everything to one side of the sweep line is completely solved. For example, to find the intersection of a number of line segments, the segments end points can be sorted into a queue first by their y coordinates, then by their x coordinates for those points with equal y coordinates. The sweep line starts at the top left most point and steps successively between each point in the queue. Status structures are maintained at each step, and operations are performed to add and remove items from the structures. The sweep line algorithm is designed in such a way that for all positions of the sweep line, all intersection points above the line have been found.

E.3. Basic algorithm

The algorithms reviewed perform boolean operations on two polygons. In order to use these algorithms to find the union of an arbitrary number of polygons, they need to be performed multiple times using a strategy such as divide and conquer; that is, find the union of sets of two polygons, then find the union of sets of two of those resulting polygons, and continue until only one polygon remains. The fact that the problem as stated only needs to find a union and does not need other boolean

operations, can then be exploited to operate on an arbitrary number of polygons.

The POLYGONUNION algorithm shown in figure E-32 and presented here is a sweep line algorithm that basically follows the FINDINTERSECTIONS algorithm from de Berg et al. (1997, p. 25), with a preprocessing step to label the edges of the polygons as being on the left or right side, and a step to determine if an edge is on the boundary of the union polygon. The significant and, it turns out, extremely easy part of the algorithm is the method for determining the boundary edges. If a horizontal line representing a sweep line intersects the subject polygons, a running count of how many polygons overlap at a specific point on the line can be kept and stored for each edge, as shown in figure E-33. This count, hereafter called *tag*, is determined when an edge is inserted into the sweep line status (when the sweep line reaches the top of the edge), and is always determined by the tag of the edge immediately to the left of the inserted edge. If there is no edge to the left, hereafter called *left edge*, as with inserted edge 1 in figure E-33, the tag for the inserted edge is set to one. If the labels of the inserted edge and left edge are different, as with edges 5, 8, 9, 11, and 13, the tag remains the same. If the labels are both left, as with edges 2, 3, 4, and 12, the tag is incremented. Finally, if the labels are both right, as with edges 6, 7, 10, and 14, the tag is decremented. Any edge with a tag value of one is on the boundary of the union polygon.

The general position assumption is:

Algorithm POLYGONUNION(P)*Input:*

A set $P = \{p_1, \dots, p_m\}$ of m polygons in general position with a total of n vertices

Output:

A set B of edges that bound the union of P

1. **for** each polygon p_i in P
2. MAKEANDLABELLEDGES (p_i, E)
3. Initialize an empty queue structure Q .
4. Initialize an empty status structure T .
5. Initialize an empty boundary edge result list B .
6. **for** each edge e_i in E
7. Insert a top event into Q corresponding to the top vertex of e_i .
8. Insert a bottom event into Q corresponding to the bottom vertex of e_i .
9. **while** Q is not empty
10. Get the next event v in Q
11. **if** v is a top event
12. HANDLETOPEVENT(v, Q, T)
13. **else**
14. HANDLEBOTTOMEVENT(v, Q, T, B)
15. **return** boundary edge list B .

Fig. E-32. Algorithm to compute the union of a number of polygons

- Each polygon is simple; that is, non-self-intersecting and without holes.
- Polygon vertices are listed in counter-clockwise order.
- Horizontal segments are allowed, with the left vertex considered the top vertex.
- Co-linear edges can overlap.
- Output is not stitched.

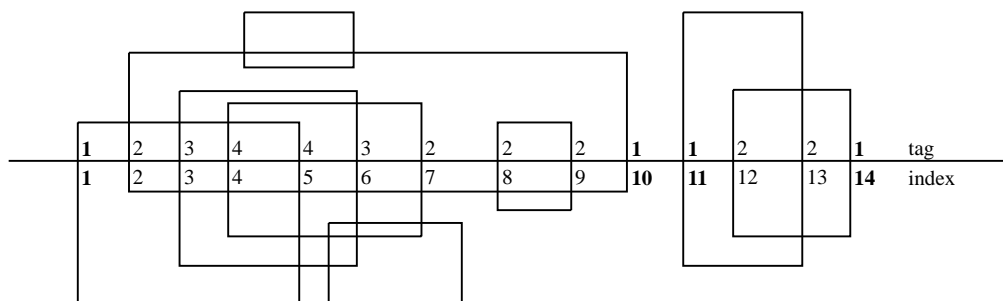


Fig. E-33. Edge tag values at a sweep line

An *edge* structure contains pointers to the vertices defining its endpoints, a label defining whether or not it is on the left side or the right side of the polygon it bounds, and a tag that will eventually define how many polygons overlap at that edge.

An *event* structure contains a pointer to the edge associated with it, a label defining if it is a top or bottom event, and a method for determining the relative order between two events. Each edge produces two events, one for the top vertex and one for the bottom. Each intersection point contributes at most four events. Two segments that intersect need to be split at the intersection point. Two new bottom events are produced for the two segments above the intersection point, and two new top events are produced for the two segments below.

A *queue* structure is used to maintain an ordered list of events and is represented with a balanced binary search tree. Top and bottom events are added to the queue at the beginning of the algorithm, while intersection events are added to the queue as the sweep line encounters them. The events in the queue are ordered first by decreasing y coordinate, then by increasing x coordinate for events with the same y coordinate. For events with the same x and y coordinates, bottom events come before top events. Multiple *bottom* events at the same point are ordered as if the event point was the intersection point of the edge associated with the event and a line

immediately *above* the sweep line. Similarly, multiple *top* events at the same point are ordered as if the event point was the intersection point of the edge associated with the event and a line immediately *below* the sweep line. In this way, there is a unique ordering of events.

A *status* structure is used to maintain an ordered list of edges that currently intersect the sweep line and is also represented with a balanced binary search tree. The sweep line invariant is that the tag value for all segments that have already been processed by the sweep line has been determined, and that all new intersections are below (or to the right of) the current event point being processed.

E.3.1. Edge labeling

Each edge of a polygon needs to be initially labeled as being a left edge or a right edge. The algorithm for labeling edges starts at the topmost vertex of the polygon (or if more than one vertex has the same largest y coordinate, the leftmost vertex among those). It then traverses each vertex in counter-clockwise order, creates a new edge, and determines the appropriate label for the edge. If the next vertex is lower than the current vertex (or to the right for horizontal lines), the edge between those points is a left edge. If the next vertex is higher (to the left), the edge is a right edge. The algorithm for making and labeling edges from a polygon is shown in figure E-34.

E.3.2. Event handling

Referring to figure E-35, at a top event, an edge that first encounters the sweep line is added to the status structure, it is tested for intersection with its neighbors on the sweep line, and its tag value is determined.

Figure E-36, at a bottom event, a boundary edge may be added to the result list, an edge is removed from the sweep line status, and new neighboring edges are

Algorithm MAKEANDLABELEDGES (p, E)*Input:*A polygon p with a set $V = v_1, \dots, v_n$ of verticesAn edge list E *Output:*A set of labeled edges for p , inserted into E .

1. Find the topmost(leftmost) vertex v_t in V .
2. **for** each vertex v_i in V , starting at v_t
3. $v_n =$ the next vertex in counter-clockwise order from v_i
4. Make a new edge e with vertices v_i and v_n and add it to E .
5. **if** $v_n.y < v_i.y$
6. $e.label = LEFT$
7. **else if** $v_n.y > v_i.y$
8. $e.label = RIGHT$
9. **else** (the edge is horizontal)
10. **if** $v_n.x > v_i.x$
11. $e.label = LEFT$
12. **else if** $v_n.x < v_i.x$
13. $e.label = RIGHT$
14. **else** (the edge has zero length)
15. Remove e from E .

Fig. E-34. Algorithm to make and label edges from a polygon

tested for intersection and handled.

When two segments intersect, each is split into two new segments, and new events are added to the event queue. Figure E-37 shows the basic algorithm for determining if two edges intersect. It does not show the geometric process of finding the intersection coordinates, but shows that the intersection test only occurs for edges from different polygons, and the return value that indicates intersection or

Algorithm HANDLETOPEVENT (v, Q, T)*Input:*An event v with associated edge e *Output:*A tag value for edge e New events that might result from edges intersecting with e

1. Insert e into status structure T .
2. Get edges e_l and e_r , the edges immediately to the left and right of e in T , respectively.
3. **if** EDGESINTERSECT(e, e_l, p)
4. HANDLEINTERSECTION(e, e_l, p, Q)
5. **if** EDGESINTERSECT(e, e_r, p)
6. HANDLEINTERSECTION(e, e_r, p, Q)
7. SETTAG(e, e_l)

Fig. E-35. Algorithm to handle a top event

not. Figure E-38 shows the algorithm for handling two edges once it is determined that they intersect.

E.3.3. Bounding edge determination

As noted earlier, the most significant part of the algorithm is the one that determines the boundary condition of each edge; that is, whether or not the edge is part of the boundary of the union polygon. It turns out that this algorithm is exceedingly simple. As the sweep line encounters each edge, the tag value of the edge immediately to its left completely determines its boundary condition. Figure E-39 shows the algorithm for setting the tag value of an edge just added to a status structure. It would help to understand the algorithm to follow along the status line in figure E-33.

Algorithm HANDLEBOTTOMEVENT (v, Q, T, B)

Input:

An event v with associated edge e .

Output:

Edge e potentially added to boundary result list

New events that might result from edges intersecting with the edges on each side of e

1. **if** $e.tag = 1$
2. Add e to boundary edge result list B .
3. Get edges e_l and e_r , the edges immediately to the left and right of e in T , respectively.
4. Remove e from T .
5. **if** EDGESINTERSECT(e_l, e_r, p)
6. HANDLEINTERSECTION(e_l, e_r, p, Q)

Fig. E-36. Algorithm to handle a bottom event

E.4. Overlapping edges

The general position condition that no two parallel edges overlap can be relaxed by modifying the EDGESINTERSECT and HANDLEINTERSECTION functions. Overlapping edges need to be considered valid intersections in EDGESINTERSECT, but need to be handled differently in HANDLEINTERSECTION. Overlapping edges with different left/right labels, as shown in figure E-40a, need to be removed from the event queue. If they are not removed and have a tag value of 1, they may be returned as boundary edges when they should not be. Overlapping edges with the same label, as shown in figure E-40b, need to be added to the event queue, and then handled normally. In this example, only one of the overlapping segments will have a tag value

Algorithm EDGESINTERSECT (e_1, e_2, p)*Input:*Two edges e_1 and e_2 *Output:*Intersection state of edges e_1 and e_2 *true* if they intersectThe intersection point p *false* if they do not intersect

1. **if** e_1 and e_2 are part of the same polygon
2. **return** *false*
3. **if** e_1 and e_2 intersect
4. Set p to be the intersection point
5. **return** *true*
6. **else**
7. **return** *false*

Fig. E-37. Algorithm to determine if two edges intersect each other

of 1 and returned as a boundary edge, while the other will have a tag value of 2 and is needed to balance the lower left edge of the smaller rectangle. (This example also demonstrates the potential problem of this solution by allowing co-linear boundary edges. The three bottom right segments are co-linear and should be returned as a single segment, in order to produce a correct polygonal structure.) Instead of simply returning *true* or *false*, EDGESINTERSECT then needs to return either *false*, *normal*, *overlapSame*, or *overlapDifferent*. For both of the *overlapSame* and *overlapDifferent* conditions, HANDLEINTERSECTION should split each edge into two segments using an endpoint from the other edge. If the return value is *overlapSame*, two new edges and four new events are created. If the return value is *overlapDifferent*, the

Algorithm HANDLEINTERSECTION (e_1, e_2, p, Q)

Input:

Edges e_1 and e_2 that intersect each other

Point p where they intersect

Output:

New events resulting from creating new edges

1. **if** e_1 's endpoints are different than p
2. Split e_1 into two segments by making a copy of it, changing both the bottom point of e_1 and the top point of it's copy to point to p .
3. Add a new bottom event to Q associated with e_1 .
4. Add a new top event to Q associated with the copy of e_1 .
5. **if** e_2 's endpoints are different than p
6. Same as lines 2-4, but with e_2 .

Fig. E-38. Algorithm to handle the intersection of two edges

overlapping segments are deleted and not added to the event queue.

The ability to handle overlapping edges was implemented in the prototype software application, similar to that described above. The implementation is not described in detail here because it is boring and tediously complex.

E.5. Computational Complexity

As stated previously, this algorithm is essentially a modified version of the FINDINTERSECTIONS algorithm described in de Berg et al. (1997, p. 25). The significant additions are the MAKEANDLABELLEDGES and SETTAG functions.

MAKEANDLABELLEDGES is run on line 3 of POLYGONUNION for each of the input polygons. Its running time is based on the number of vertices of each polygon.

Algorithm SETTAG (e, e_l)*Input:*An edge e The edge e_l immediately to the left of e when it was
inserted into T *Output:*The tag value for e

1. **if** e_l does not exist
2. $e.tag = 1$
3. **else if** $e.label \neq e_l.label$
4. $e.tag = e_l.tag$
5. **else if** $e.label = LEFT$
6. $e.tag = e_l.tag + 1$
7. **else** $e.label = RIGHT$
8. $e.tag = e_l.tag - 1$

Fig. E-39. Algorithm to set tag

Its total running time is then based on the total number of polygon vertices, which is n . So lines 2 and 3 take $O(n)$ time.

SETTAG runs in constant time, so does not affect the asymptotic running time of its calling function, HANDLETOPEVENT.

Line 10 in POLYGONUNION runs for each event. The total number of events is two times the number of input vertices, $2n$, plus four times the number of intersection points, $4k$. So line 10's running time is $O(n+k)$. Line 11 in POLYGONUNION, and the functions HANDLETOPEVENT and HANDLEBOTTOPEVENT all perform operations on balanced binary search trees, so their running times are $O(\log n)$.

So the running time of POLYGONUNION is $O((n+k) \log n)$, where n is the total

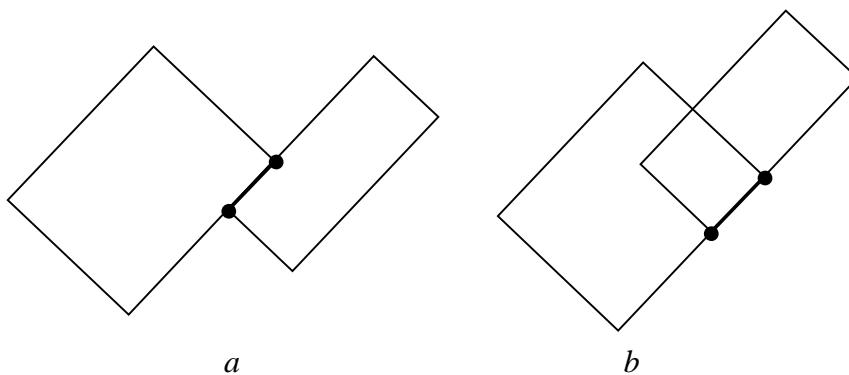


Fig. E-40. Overlapping edges

number of polygonal vertices, and k is the total number of intersections between edges from different polygons.

E.6. Extensions

The general position condition that polygons cannot be self-intersecting can be relaxed if the `MAKEANDLABELEDGES` function is modified to find and handle intersections between edges. Once an intersection point is reached, the labeling tests would have to be switched from assuming counter-clockwise vertex ordering to clockwise ordering.

The general position condition that polygons cannot have holes can be relaxed if the `MAKEANDLABELEDGES` function is modified to expect a polygon data structure that allows for holes in its representation. Changes similar to those just noted can be made to appropriately label hole edges.

E.7. Discussion

The advantages of using this algorithm over ones designed for other boolean operations are:

- it can handle an arbitrary number of polygons,
- it requires no preprocessing step (other than the labeling step at the beginning, which is making explicit a property of the parts of a polygon),
- it does not require any polygon based geometric operations, and
- it is conceptually much simpler.

This approach of using labels and analyzing edges as they relate to each other on a sweep line can probably be extended to handle the other boolean operations on two input polygons. However, many other algorithms exist that perform this function, and it will probably not be much more advantageous, if at all.

VITA

Scott Anthony Arvin

39 Summit Dr.
New Boston, NH 03070

Education

Ph.D. in Architecture	Texas A&M University	5/04
M.S. in Management	Troy State University	9/91
B. Arch.	University of Notre Dame	5/84

Awards and Achievements

Mitchell Fellowship in Construction Management	9/99 – 5/00
College of Architecture Research Grant	6/99 – 8/99
Mitchell Fellowship in Construction Management	9/98 – 5/99
Caudill Graduate Research Fellowship	9/96 – 5/97
Lechner Fellowship	9/94 – 5/95
Architectural Registration, Florida	3/98

Research Projects

Energy Resource Program	9/95 – 9/96
Dixie National Forest Animation	5/95 – 10/97

Employment

Autodesk	Software Engineer	7/00 – present
Texas A&M University	Graduate Assistant	9/98 – 5/00
	Assistant Lecturer	9/96 – 5/98
Foil Wyatt Architects	Intern Architect	5/92 – 7/94
United States Air Force	Engineering Liaison	7/89 – 8/91
	Chief of Readiness	7/88 – 7/89
	Architect	10/84 – 7/88