EXPLORATIONS OF COMPUTER-BASED DESIGN TOOLS FOR URBAN DESIGN PROJECTS

by
Joseph M. Knight
B.Arch., University of Tennessee
Knoxville, Tennessee
June 1986

Submitted to the Department of Architecture
in partial fulfillment of the requirements for the Degree of
Master of Science in Architecture Studies
at the
Massachusetts Institute of Technology
June 1992

Signature of the author _____
Joseph M. Knight, Department of Architecture
May 12, 1992

Certified by _____
William L. Porter
Muriel and Norman Leventhal Professor of Architecture and Planning
Thesis Supervisor

Accepted by _____
Julian Beinart
Chairman, Departmental Committee on Graduate Students

Rotch

EXPLORATIONS OF COMPUTER-BASED DESIGN TOOLS FOR URBAN DESIGN PROJECTS

by
Joseph M. Knight

Submitted to the Department of Architecture on May 12, 1992
in partial fulfillment of the requirements for the Degree of
Master of Science in Architecture Studies

ABSTRACT

This thesis is an investigation into issues involving computers, information, and automation in the designing of large-scale environments. It is an attempt to understand the issues at root in developing an "intelligent" design environment which provides tools for handling tasks often too mundane and distracting for sustained design activity. In the process of devising these tools, fundamental issues regarding the elements or objects of design, their characteristics, and the transformations they undergo are revealed in light of the particular capabilities of the computer.

This study was undertaken as an attempt to discover these issues for myself by working to create a system of tools on top of an existing computer-aided design program - a personalized design environment. The path of discovery taken is reconstructed in this paper, in part to illustrate some of the pitfalls of dealing with real-world programming tasks, and also to demonstrate the revelation of the inherent issues that are involved when attempting such a project. Although the programmed end-product is incomplete and greatly simplifies the true nature of such a design problem, the lessons learned are distilled and clarified to provide a basis for further work and investigation in this field.

The first part is a synopsis of issues related to computer-aided design: an historic overview, current applications, on-going research and forecasts. This section is provided to illustrate the foundation of understanding that I had when undertaking to develop tools of my own. The second part includes the initial tool concepts and their intended purposes, a discussion of hardware and software platforms, and numerous considerations I was compelled to address while developing modeling and information-handling components of the design environment. Part three deals in depth with a sophisticated tool proposal for instantiating urban type elements for illustrating a possible "realization" of a schematic design; this tool was not able to be developed on the chosen platform. I have also included some possible scenarios for using the modeling and information handling tools.

Thesis Supervisor: William Porter
Title: Muriel and Norman Leventhal Professor of Architecture and Planning

EXPLORATIONS OF COMPUTER-BASED DESIGN TOOLS FOR URBAN DESIGN PROJECTS

Contents

"Standing on the shoulders of giants leaves me cold..."
Berry, Buck, Mills, Stipe

Introduction - The Nature of this Study

A Personal Investigation of CAD Issues

Having initiated my higher-level education with an eye on research science, but then abandoning that to pursue the study of architecture from a humanities and design point of view, I feel my intellectual leanings tend to straddle the vast frontier of art and science.

As an architectural intern, I managed to work myself into positions of design responsibility, which I would tackle with a ruthless rationality, often impatient with the whimsy and biases of more experienced architects. At the same time, I took advantage of every possible opportunity to use computer-aided design (CAD) systems in the offices where I worked. Almost exclusively self-taught on the three systems I came into contact with over those four years, I tended to utilize them in creative and unorthodox ways, producing results far beyond my superiors' expectations. When the time came, I was the obvious choice for office CAD manager. I was not satisfied with the responsibilities of managing drawing production and output, and spent a great deal of time finding new innovative means of using these exciting tools for the benefit of the firm. Eventually I felt the need to formally educate myself to the issues of computers and architectural design. I applied to and entered M.I.T. with high hopes and bold dreams to fuse the art of design with the science of computational procedures.

What I found once I had immersed myself into the fray of the Computer Resources Laboratory (CRL) in the Department of Architecture was the realization of CAD as an issue of impressive, and daunting, scope and

complexity. I recognized in the work of some kernels of some ideas I had had already, and in others', strong refutations of other ideas I hoped to pursue. The prevailing controversy and disagreement about the very nature of design and its counterpart in machine was somewhat discouraging. Nevertheless, I also saw a wide-open frontier, the resolution of which would be long in coming but would completely revolutionize the design practices, hopefully within my own lifetime. I wanted to be a part of that revolution.

During my time at M.I.T., I took a mix of courses in computer science and applications and in urban and environmental design. I felt that if I focused my work on larger-scale design issues (where I've had a long-lasting, latent interest), I might circumvent some of the intractable issues inherent in the study of computer applications in architectural design. I attempted in two cases to utilize the technology available at the CRL through my computer courses in the development and presentation of projects in my urban design courses. These projects have proven very useful in educating me on some of the issues of presentation and visualization with computers. Either could have been expanded to a thesis project, but I felt the need to tackle the issue of the processes of design and how the computer could assist a trained designer to produce better designs. In this regard, the Design Research Seminar at M.I.T has proven very useful to understanding some of the issues at hand.

Goals of this thesis

My best-learned lessons begin with rejection of established rules and norms, followed by usually painful lessons of experience. Unfortunately, I find this is the best way for me to truly understand an issue. Having studied a number of approaches to CAD development, both theoretical and applied, I found my understanding of the problems as defined in these studies still to be limited. I questioned many of the assumptions that I encountered. I thought that I might give the issue my best shot.

Only through direct experience in addressing the questions of computers and the design process did I feel I could gain a significant understanding of the problems and pitfalls, and perhaps of some of the opportunities. Therefore the primary personal goal of this thesis was to freely explore some of the many ideas and assumptions I have already had, to generate some new ones in the

process, and to tackle the programming tasks they would require. Perhaps more importantly, I have found that by examining more specifically some of the issues involving the processes of design, things which I'd had much experience with but little applied understanding, and by working on real programming problems, some of the fundamental issues at hand reveal themselves resoundingly. This process of discovery I would qualify as the academic goal of this work. Finally, I feel that this thesis would represent a kernel for work I hope to pursue later in this highly promising field.

The working methodology of this thesis

I have chosen a working method which best suits my personal style: a "can do" style of creativity and learning from experience, rather than through scholarly research and synthesis of pre-existing written authority. My focus, therefore, is on a *project* rather than on a *subject*.

What my thesis attempts to do is explore, quite independently, what I might do to enhance an existing CAD program to better assist a designer to develop a plan; the best method I found for such an exploration is to *just do it* . This best suits my personal learning capacities: revelations through problem solving. Throughout the entire process I attempt to be as explicit as possible about my assumptions and directions. At the risk of being incorrect, I will have the advantage of continually moving forward. From this explicitness, I will in conclusion gain valuable insight from the successes and failures produced.

The thesis product is a number of procedures, written in a variation of Pascal language, which manipulate objects and information in ways particular to an urban design problem. The point is to automate to the best of my abilities some of the rote work regarding modeling, information management, and visualization encountered during a design session.

The first part of the thesis paper is a synopsis of issues related to computer-aided design: an historic overview, current applications, on-going research and forecasts. This section is provided to illustrate the foundation of understanding that I had when undertaking to develop tools of my own. The second part includes the initial tool concepts and their intended purposes, a discussion of

hardware and software platforms, and numerous considerations I was compelled to address while developing modeling and information-handling components of the design environment. Part three deals in depth with a sophisticated tool proposal for incorporating urban type elements into a schematic model for illustrating a possible "realization" of the schematic design; this tool was not able to be developed on the chosen platform. I have also included some possible scenarios for using the modeling and information handling tools.

Part 1 - A General Summary of Ideas Concerning Computers and Designing

Computer Aided Design: an Overview

At the advent of the much-heralded "Information Age", the computer has become more pervasive and influential than ever imagined 30 years ago. There are few remaining aspects of our daily lives that are not in some way influenced, controlled, or otherwise made possible through the use of computers and information technology. This is especially true in the workplace, where the business or firm that does not use even the most basic of word processors is considered an anachronism. Business quickly learned the benefits of the computer's speed and accuracy to enhance production. Modern manufacturing and industry have made great use of computers and robotics to speed production and lower costs.

While the computer has found wide acceptance in many areas of human endeavor, there remain many domains where the utility of the computer has been suspect. From the beginning, there have been fears of the computers replacing humans in the workforce, rendering them obsolete; this has proven unfounded, for new and better occupations in the information and service industries have been created to fill the gap. What is it about human abilities that the computer has been unable to duplicate? It has much to do with human ingenuity and imagination, and the ability to make intuitive leaps in associations and knowledge beyond what programming rules can dictate. This conflict is most clearly demonstrated in the development of computer aided design (CAD), and in its successes and failures.

Origins of Computer-Aided Design

In the early 1960's initial experiments were made using computers for design purposes. Ivan Sutherland developed Sketchpad at M.I.T.'s LIncoln Laboratory in 1963, which utilized a TX-2 computer to represent primitive two-dimensional shapes on an interactive graphics terminal, allowing manipulation of these shapes in real-time by a user. This particular set-up required what at the time was an extraordinary amount of computer power. Nevertheless, these experiments laid down some fundamental rules concerning computers and graphic manipulation. By the end of the decade, several automobile and aerospace companies had developed and implemented ever more sophisticated interactive systems for engineering design.

Computers in Architecture

These tools remained in the hands of very large industrial and engineering companies for many years, for they were the only interested parties capable of affording the very large and expensive systems required. Few architects were eager to turn their profession over to machines anyway. However, the technology continued to improve, and in the 1970's, with the proliferation of the integrated circuit and the microchip, some architecture firms were able invest in systems of their own, primarily for automating production of construction documents. With the introduction of the personal computer (PC) in the 1980's, computers became an overwhelming trend in architecture firms. (Mitchell 1977)

Critical to the proliferation of CAD in the architectural office was the available, appropriate software. So-called first generation CAD software applications were simple drawing tools, an "electronic pencil", useful for editing and automating some aspects of drawing production. It was soon realized that these drawings could become much more functional and therefore valuable if the drawing elements could be linked to information regarding what they actually represented. With the addition of attribute files and symbol libraries and their associated spreadsheets, a fair degree of automation could be achieved in the production of construction documents. Thus it is today that the

great majority of commercially available CAD software incorporates this level of utility and little more. (Pohl & Myers 1992)

Recent Developments in Computer Applications to Architecture

However, the ever increasing available speed and power for the typical office is now reaching a threshold where much more far-reaching and influential tasks are being explored commercially. A great deal of interest is currently focused on techniques of visualization. Once relegated to only the very large firms and their high-powered hardware, now any office can now afford a system that can produce high-quality representations of design proposals. Software has progressed from allowing only two-dimensional drawing to sophisticated three-dimensional modeling with shading and shadowing, through ray-tracing, and into radiosity as means of producing ever more "photo-real" images, each level requiring significantly more processing power.

More significantly, I believe, developments coinciding with available power have led to a greater emphasis on user interface, providing easier and more intuitive means of issuing commands and generally interacting with the software. The Apple Macintosh standard of graphical user interface has opened the way for other platforms to create CAD systems that almost anyone can learn with little effort. Very recently, "pen-based" interfaces have been introduced, which allow for input and feedback directly to drawing surfaces and notepads, with full command options and sketch-mark and notation recognition. Even voice recognition of commands will soon be commercially available, providing another alternative to the usual mouse/screen interface. The progress these technologies make towards putting CAD into the hands of *everyone* in the design office, and not just those of the CAD draftsman or operating specialist, will have much more far-reaching significance than improved image production. (Crosley 1991)

Another advance towards more general and available CAD systems is system integration. Whereas computer technology has been used primarily for creation and presentation of designs, improved networks and translation capabilities will allow for a much more efficient transfer of design information between specialties: to engineers, product suppliers, public agencies for code

enforcement, etc. Even some clients are now demanding from architects as-built data regarding their facilities for use in their own formats. These advances will greatly reduce the costs and errors associated with manual transfers of information. (Lischewski 1991)

Although computer applications in engineering have long outpaced advances in architecture, now we are seeing more and more architecturally-specific engineering applications that are available for use by architects. Such "expert-systems" greatly reduce reproduction and iteration of designs between disciplines and help to produce more integrated solutions. Such applications include systems for structural and energy analysis, code checking, cost estimating, and HVAC and plumbing aids.

CAD Frontiers

Fundamental to recent advances in CAD capability is the growing need to move beyond the "electronic pencil", or merely high-tech drafting, and to begin using the computer to develop virtual models of a design proposal, with all the qualities and characteristics of a real product. This means that a computer representation of a steel beam, for example, must include in addition to its three dimensional existence qualities of its strength, cost, etc. This imbuing of elements in a CAD model with intelligence or self-knowledge is *object-oriented* in focus, no longer relying on collections of line primitives entirely subject to interpretation by the designer and builder. With such tools at hand, the designer has much greater ability to experiment and test real alternatives at much, much less risk and expense than on the finished product.

There are major research efforts underway to develop CAD systems which do far more than merely record and communicate design proposals. In these cases the goal is to provide comprehensive *design assistance* to handle many of the regular tasks of the designer for him or her. Whereas many of the tools discussed already assist the designer at the task, the true design assistant would also handle a great deal of the *reasoning* that the designer must perform. The goal of these systems would be to handle much of the information that normally inundates a designer at work by employing an array of expert systems

which would continuously analyze the current state of a design, making suggestions of alternatives to be resolved in part by an automated managing module and in part by the designer. By effectively handling much more information than before, the designer can originate a solution that is therefore more highly developed and effective. (Pohl & Meyers 1992)

Critical to the success of such a comprehensive system is the utilization of artificial intelligence. A.I. would be necessary not only for recognition and comprehension of the design state, which most often exists in a form never before seen, but also for enabling new an alternative patterns of reasoning. Such a system without A.I. would be severely restrictive, and certainly not worth pursuing. Artificial intelligence would also serve a great purpose as a mechanism for *learning*, acquiring knowledge not only in the domain of the expert systems, but also for gaining design expertise and design knowledge from experience with a user. Such a system, by learning the habits and methods of an individual designer would be able to anticipate and suggest design moves under similar circumstances on different design problems. This learning would also greatly improve the value of a system by making all design experiences and situations lasting assets to a firm. (Pohl & Meyers 1992)

Design activity and the computer

Early CAD Efforts

In computer-aided design, as in computer applications of any sort, the issue of appropriate software has always been at the forefront of interest in the expanding use of computers. The original UNIVAC computers of the 1950's, while an exceedingly expensive initial cost, became an even more enormous financial burden to use, requiring many, many hours of servicing and programming for even the most mundane tasks. The software simply was not keeping up with the hardware. With the introduction of more friendly computer languages such as FORTRAN and COBOL, the profession of computer programmer took off, and operating costs at last became manageable. With the introduction of third, fourth, and even fifth generation programming languages, mundane tasks are becoming ever simpler to program.

So it was with CAD that for many years utility was severely restricted by software limitations. The first uses amounted to no more than an "electronic pencil" for creating and editing graphic images, with the later additions of attribute handling routines. That most CAD software commercially available through the 1980's did not provide much more functionality than this is understandable, given the surprising intractability of design automation and the need for software development companies to release usable products in a timely manner.

The Search for a Model of the Design Process

Nevertheless, interest in the potentials of CAD grew quickly in the academic community in the 1960's. (Mitchell 1977) In parallel with the evolution in programming, the field of design studies was developing in new ways and informing the work of the programmers. A great deal of thought and effort were expended at the time to develop a general model of the design process which was expected to be mathematically adapted to the computer. As this task proved to be quite complex, the process was reconsidered as a collection of delegated sub-processes which would incrementally transform a statement of a design problem through a series of narrow solutions into a general solution. In the 1970's, numerous studies and models were made, each attempting to solve particular aspects of design transformation quantitatively. (Pohl & Myers 1992)

By the 1980's the growing awareness of the ill-defined nature and conflicting characteristics of design problems took hold; it now seemed that there would be no way to describe what happens during designing in purely rationalistic, procedural terms. Each design problem seemed to have its own set of rules contingent upon the specific situation. (Schoen 1988) Furthermore, each designer had his or her own unique methods and priorities. It became clear that designing simply "isn't always anything; it isn't always linear, nor always non-linear, or hierarchical, or non-hierarchical, etc....Usually the goals, consequences and specifications are not entirely known at the beginning, often conditions and criteria evolve and may be discarded, so the process of testing, or searching, has no predictable outcome." (Ervin.) Therefore, no general model could exist, and that these algorithmically-based approaches would never provide more than a secondary role. In fact, the design process is now

considered not a process at all, but a matter of behavior and subjective interpretation, based in cognitive and psychological process that are poorly understood. The current term, more appropriate though more ambiguous, for what happens during design is "design activity." (Pohl & Myers 1992)

Design Worlds

One of the more recently influential descriptions characterizing design activity has been provided by Schoen and explicitly laid out in "Designing: Rules, Types, and Worlds" (1988). Here, Schoen identifies some of the paradoxes of design knowledge and activity: the inability of designers to explicitly describe the knowledge they hold and employ; the utilization of general rules yet deriving unique solutions; the importance of accumulated prototype knowledge yet the continuing generation of new prototypes; and the ability of professionals of different backgrounds, and therefore different priorities, to agree on a common solution. These raise many questions regarding how reasonable and rational solutions can arise out of all of these conflicting activities.

To help explain, Schoen speaks of designers entering into "design worlds", an environment of ideas, concepts and knowledge where the design problem at hand carries its own rules and objects for consideration and experimentation. Such paradoxes as mentioned would be allowed here, and dealt with. Mobility of thinking within this world is vital to the designer, who continually makes leaps in perception and knowledge associations, from one *design state* to another. A design state may be considered one of any number of alternative ways of framing or looking at a problem or design situation. The setting of the problem is critical to how subsequent design moves are made, and goes hand in hand with the solution, overlapping. A design state may be considered many different ways: functionality, spatial gestalt, or experiential, (Schoen 1988), or from the point of view of the planner, designer, or developer (Fargas 1991), or as the creator/*generator* or the evaluator/*tester* (Ervin).

The triggers for these leaps in design states are entirely unpredictable, but often are inspired by associations perceived between some aspect of the current design and prototype knowledge that the designer has accumulated. Such leaps may also be related to emotional or environmental stimuli or any

combination of such events. Whether psychological of physiological causes, it is the in the *act* of designing and in design awareness, the "design world," that these intuitive leaps are made. (Schoen 1988)

The Role of Rules and Types in Designing

While researchers were attempting to model the design process, they found it necessary for them to determine consistent *rules* to be obeyed, assuming that design was a rule-based system. As it turns out, there are virtually no consistent rules which would apply to all design situations. As Schoen suggests, design rules are "almost always treated as contingent and contextual. They are held tentatively, always admitting exceptions." (1988) In light of this, how could any rules be applied to design in a computer design system?

To Schoen, the answer is found in another critical aspect of the design world: types, from which rules are derived. Types, or *prototypes*, are pre-existing examples of solutions ("exemplars, precedents, images, and concrete universals") held in the memory of the designer and drawn upon as design-knowledge references in later design problems. Arnheim calls them "generative abstractions", simplified, non-specific categories that have the "fullness" of particulars. Schoen goes on to define four different types of types: functional, reference, spatial gestalt, and experiential archetypes. (Schoen 1988) Prototypes may exist for any scale object, from column capitols to capital cities.

Only when considering a type in the realm of a design problem do rules emerge, specifically related to that type and applicable to that problem. One could say that the designer exercises reasoning with *objects*, representing *types*, with the types' particular rules implicit or explicit. When another object/type is considered, a whole new set of rules will need consideration and within the context of the already considered object/types. Gradually, as the designer moves through the "design world", considering different types in different design states, incorporating some notions, discarding others, and varying still others, the various rules derived are gradually satisfied and assimilated in the design, and "fixed."

Computational Design Assistance

Given the complexity of design activity, there probably will never be a complete computer-designer as imagined 30 years ago; the human designer has proven to be critical to the activity. Instead, what is in order is an assistant or partner to the designer; a tool really, but of very high sophistication and utility. Much research activity lately has been in modeling an *intelligent* CAD system, one which is highly flexible and adaptive to the designer and the design situation. (Pohl & Myers 1988, Gross 1985, Hillier, Habraken 1985)

One may wonder at what points in the activity that automation is useful. Some work has shown that the computer may be useful during the 'generator' phases of a design, grinding out a variety of solutions. (Habraken) On the other hand, it may also be helpful for testing a proposed solution, for compliance to rules etc. by using a *constraint manager.* (Gross, Ervin, Fleisher 1987) But the computer should not and cannot be delegated both tasks at once. Rather, the cycle between generating ideas and testing them are not at all clear cut, and is often short-circuited in practice whenever the designer makes unexpected leaps in association or recognition in the design state. I would assert that when and where to use computational aids should be solely the at discretion of the designer, to benefit and not hinder his or her intuitive understandings and actions.

In answer to the question of types brought to a design problem, it is argued that the computer is ably suited to the task of compiling information about a variety of prototypes, to be accessed and utilized by the designer. Whereas the number and detail of such prototype data is nearly limitless, it has been suggested that a designer would build up an individual *repertoire* of ones s/he finds relevant and useful from experience. The issues of how this repertoire is compiled and how it can be accessed intuitively in the heat of the design activity are currently being addressed by methods of artificial intelligence. There are studies of how artificial intelligence can be utilized to customize preferences of representations of such design knowledge, as well as anticipate many of the associative leaps that a designer makes while designing. (Pohl & Myers)

In light of the magnitude and scope of the on-going studies in intelligent CAD systems for design, I found it difficult to find a particular avenue of research which suited my own temperament. As primarily a designer, I felt that by involving myself in just one of the many particular details of this task would probably cause me to lose sight of my simple goal: to facilitate or automate some of the rote tasks a designer regularly encounters, thereby saving time on any one proposal, and increasing the number of alternatives that can be considered practically. Rather than spending the bulk of my thesis study examining the fundamental nature of these tasks, I felt that I should just jump right in and see what I would do, simply and with limited resources. By doing so I would expect to discover from my own experience the computational principles that can effect designing. I would at the very least gain a greater appreciation of the task, and I would perhaps demonstrate a few insights on the problem.

Computer-Aided Urban Design - a Logical Extension

By focusing my study on larger-scale design problems, namely urban design or environmental design (involving the planning and positioning of buildings and spaces), I hoped for several things: to avoid in part some of the intractable problems associated with architectural design, to perhaps take advantage of some of the work already done in the more rational field of urban planning, and to address design problems currently of interest to me.

A Link Between Architecture and Planning

Environmental design, as it has sometimes been defined at M.I.T., can be seen as a link between the fields of architecture and urban planning. (I would prefer a non-suggestive phrase such as "design of large-scale developments") Architecture deals usually with a specific building or structure and with single ownership and client. The architectural designer exercises his or her craft with relative autonomy and subjectivity, often resorting to theoretical social, spatial, and style issues. The urban planner attempts to deal more in the realm of scientific rationality, handling quantitative issues for resource allocation and forecasting. The planner's domain usually encompasses large areas with multiple ownerships, and with a client representing the public interest. (These

characterizations are intentionally generalized and exaggerated.) Both architecture and planning deal with issues of space, but in very different terms and with different agendas.

I see the study of design at the "urban" or "environmental" level to be a mediator between urban planning and architectural design, as the link between scientific abstraction and intuitive human interpretation. Actually, urban designing is probably the more suited than architecture to computer-aided design and modeling. The spatial constructs are more simple, and the information intensity level tends to be much higher, abetted by developments in computational urban planning methods. Designing at the urban scale is a very different thing from designing a building; in architectural design, the designer is trying to fit elements (spaces) into a given volume, or making a fit such that those spaces determine the character and expressiveness of the volume or facade. At the scale of urban design, I see the design elements to be individual buildings and public spaces. The qualities and interactions of these elements are of a very different nature than those qualities worked with in architecture, involving public and private territories participating in a sort of figure/ground composition. The design modeling of this composition holds special advantages over architectural modeling.

Urban design does not explicitly exist as a specialized occupation, and is usually practiced by architectural designers or urban planners as an extension of their trained specialties. For this reason, I see a particular usefulness in such a tool as I propose. Perhaps by instilling this tool with intelligence and knowledge of basic urban design principles, it could serve in an educative capacity to students and to practitioners not expressly trained in the issues of urban and environmental design. The designer, therefore, is given a tool for modeling space and volume, for developing an appropriate morphological expression from information intensive, quantitative givens. The planner, on the other hand, is given a means to realize spatially the numerical determinants s/he has deemed appropriate. The architect is rewarded with immediate

visualization; the planner with immediate consequences. It is my hope that with such means and information provided, a more thorough and sober investigation by both may be made possible, therefore resulting in better thought-out designs and design guidelines.

Part 2 - Conceptualizing, Developing and Implementing Some Computer-Based Tools for the Schematic Design of Large Scale Developments

This section attempts to present in a reasonable format the experiences and realizations I had over many weeks as I attempted to develop some ideas for design tools into actual program components. My purpose is to share my experiences and lessons to any who may attempt a similar project. Because this thesis is a very independent and personal investigation, many assumptions have been made. It is my hope that in this section I can be very explicit about these assumptions, thereby possibly finding more widely applicable generalities about design and computing.

Conceptual Tool Proposals

Probably the most glaring assumptions I have made have to do with the nature of design activity and what I decided would be useful tools for a designer working at the schematic level. The tools that I envisioned were to be nothing more that task handlers that I would like to have as a designer; they would of course be best suited to my way of working on a design. If we can then assume that my way of designing isn't completely individual, and that there could well be some aspects that are common among designers, then the tools I would develop have some general usefulness. It should also be noted that the idea of design presented in this study is strictly *morphological*; the determination and

resolution of programmatic and functional requirements I leave to the user's expertise.

The following tool summaries were proposed before I had begun investigating available hardware and software resources, and therefore constitute a "wish list" of task handlers I imagine would be useful and helpful to an urban design situation. Still, they are quite modest in that they do not claim to incorporate any extreme measure of *intelligence* or automation. I tried even at this stage to imagine only what I felt I could realistically accomplish in the short time available.

Modeler

The first and most fundamental assumption I've made about the morphological designing of large-scale environments is that the designer could best realize and experiment in design if given tools for physically modeling volumes and spaces. By modeling with abstracted forms as one would with lumps of clay, the designer molds and shapes masses and voids to define volumes and spaces in the urban fabric. (Fig. 2.1) With a good, intuitive modeling program, one with

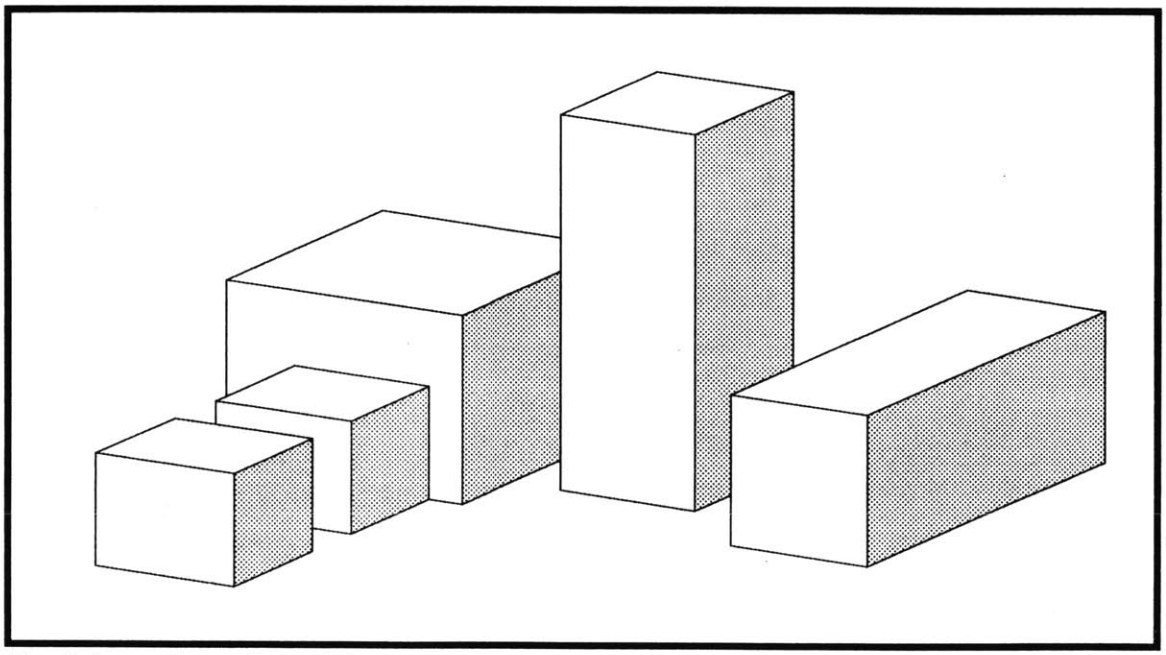


Fig. 2.1 schematic modeler

easy-to-use functions and clear graphics which do not interfere with the designer's thought processes, a user is interacting with virtual objects to create a design reality of sorts, using the abstractions and metaphors possible in geometric primitives. For the modeling component I am proposing, my concerns are for developing means of creating and editing objects specific to large-scale design tasks. I should also be concerned with what *elements* comprise an urban design and how they may be abstracted and represented for the purposes of such a schematic modeler.

Information Handler

An urban design at any given state has inherent in it much more than simple spatial relationships of forms. By having inherent information about their very natures, the objects in a design model can then engage in a more substantial dialog with the designer. The computer therefore allows one to design with information. This concept has been explored in a number of ways already; in research, there has been work exploring designing with constraints, where the objects have information regrading do's and don'ts of positions and adjacencies. (Kalay 1987) This is obviously of more use to the schematic designer, but I have yet to see any semblance of these concepts applied in a commercially available CAD program.

The information component of the tool I propose would be useful for handling the mundane tasks of measuring and calculating quantities of a particular design proposal for compliance to the program and to code and zoning restrictions. These tasks often detract from the creative mood of designing, and that to automate and make easily queryable and accessible this information would be a definite benefit to the user. The level and complexity of information would be specifically tailored to large-scale design problems, but would probably not be as sophisticated in function as the constraint-based programs developed by others. I would prefer to avoid such assertive constraints for now, leaving this sort of intelligence to the designer/user.

To avoid a potential "glut" of data, the information database of the schematic model must be queryable with a menu of standard queries made available. Data may be broken down into grosses, nets, percentage comparisons,

degrees of compliance, and per building or phase. Again, it is my goal to allow flexibility; ideally any particular queries the designer desires would be customizable.

Greater Design Realizer

Another tool I have considered that I felt would be useful I am calling a "greater design realizer" for want of a better term. With this tool, a designer and client would be able to actually *see* the potential design result while still at an early stage in the design, a more immediate means of visual testing. I have felt that the massing models usually utilized at the schematic phase of design lack a sense of scale and rely too much on the imagination of the viewer, who must attempt to see the sterile forms of the model to be real buildings and the unfilled portions to be lively spaces. Though a trained designer may be able to accomplish this (although I've known several who could not), few clients or planning professionals can do so.

A greater design realization component of a computer based large-scale design tool would automatically convert a massing model of a design in progress into a "finished" example of how the design would look when completely implemented, with many of the qualities of such a place abstractly represented. (Fig. 2.2) This is especially relevant to urban design problems, where the product is often only prescriptive, and the details of the built results usually lie outside the influence of the original designer. The details of the realization would be random applications of standards based on some sort of user-defined

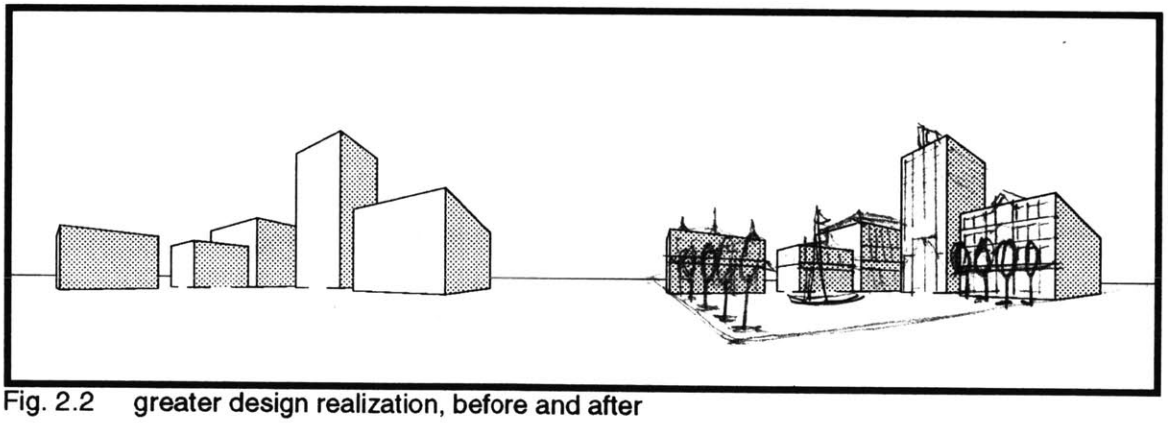


Fig. 2.2 greater design realization, before and after

prototype knowledge, or perhaps on specific guidelines of the qualities of the elements of urban design.

Consistent Visualization

One more tool component that I imagined initially would be a method of achieving consistent visual testing of proposals. Much of what makes urban designs good designs are their experiential and sequential nature. Indeed, design decisions for projects such as these are often rooted in forming a relevant, interesting, or even memorable experience or "image" of the place. For this reason, I felt that a component for consistent and logical design visualizations could be integral to the design activity, allowing for significant comparisons of alternative solutions and for charting the evolution of a particular design solution. This is a reasonable alternative to the annoying "fly-by" visualizations so prevalent these days due to the recent proliferation of affordable computer modeling and animation capabilities.

While many programs have means of saving views and viewpoints, I would imagine a system of automatically compiling libraries of views of the evolving design from various pre-defined viewpoints. These images would be automatically saved and categorized and would be a basis for more serious visual testing of the design. Various automated and queried techniques of view retrieval and slide show compilation could be employed for design evaluation and even design presentation.

(Note: Due to time constraints, I was not able to develop this last tool at all. I am mentioning it here because I feel it could be a very useful tool, and I hope that I might someday pursue this idea further.)

Hardware and software concerns and investigations

Specifications and Requirements

As I've said, the tools I proposed would be developed within or "on top of" an existing, commercially available CAD platform, to take advantage of the many

built-in functions and capabilities of the software. For handling the specific functions outlined above, certain features would be required. A 3D modeler that is intuitively easy to use would be preferred, to allow for a free exploration of form and massing. The elements in the model would need to allow for flexible attribute assigning to satisfy the basics of the information handling component. Also, a reasonable 3D viewing mode would be preferred.

The ambitiousness of these tools and the degree of automation proposed for them would probably require more than just a macro language function (mere combinations of preset commands and/or keystrokes). I would most certainly need to employ conditional statements, such as IF..THEN and the like, available only with true programming languages. Given my relative inexperience with major programming tasks, I would need a platform that is user-oriented, that is, designed to be easy to use and implement into the regular functions of the software. I would also look at specialized CAD functions provided in the programming language, at how they may aid in the development of these proposed tools.

Probable Candidates

I decided I would limit my search for appropriate hardware and software to what was available to me in the Architecture Department, whether in the Computer Resources Laboratory or in the Visual Studies Lab, to take advantage of the resources and expertise at hand. Within all of these constraints, I found two software packages to choose from: AutoCAD Release 11, on IBM-PC type platform and using AutoLISP programming language; and MiniCad 3.1+ on Macintosh with MiniPascal.

I was pretty familiar with AutoCAD, but having learned on a version now over 6 years old, I wasn't terribly familiar with its newer capabilities in 3D and attribute functions. Its 3D modeling seemed rather powerful, but not very intuitive; the same could be said for the attribute assignment components. Though I preferred LISP to Pascal for its more symbol-oriented language, I had found AutoLISP somewhat unwieldy to implement and lacking in special functions.

MiniCad had a completely different set of advantages and shortcomings. The interface, especially with customized commands, was quite good, but the 3D capabilities seemed rather primitive. The graphic interface and the object-oriented nature of the graphic elements (as opposed to the point/line elements of AutoCAD) were a definite plus. Furthermore, built-in spreadsheets and a large selection of special functions in MiniPascal made MiniCad quite attractive despite the weak 3D capabilities.

Perhaps the deciding factor was personal access to the necessary hardware. The only solution I found to guarantee myself access to the necessary resources when needed would be to invest in my own system and software. By spousal mandate, my household would have none other than a Macintosh, thereby selecting MiniCad by default as the platform for this investigation. (Note: AutoCAD on Macintosh did not yet have AutoLISP capabilities and therefore was not an option.)

MiniCad 3.1+ on the Macintosh

MiniCad is primarily a 2D graphics program with significant data incorporation techniques. It has a prevailing "smartness" to its graphics, with very clear textual data associations to objects. It also has a "smart cursor" and "screen hints," which continually search the screen for geometric associations for snapping or other guidelines, and a drawing constraints mode much more interactive and graphic than I've seen elsewhere. Though these serve no direct purpose to the tools proposed, they do seem to allow a much more interactive and accurate modeling environment, even though only two-dimensional.

In MiniCad, objects are created in 2D, and emphasis is more on surfaced polygons than on points and lines, as it has been in AutoCAD. Coloring and patterning of these surfaces are quite matter-of-fact and obvious. These aspects seem more conducive to modeling than a point and line convention. MiniCad has a conventional layering mode, but which allows a number of alternatives for viewing, snapping to, and editing one or all layers at once. Non-active layers may be "grayed" or made invisible.

Two-dimensional objects can be made three-dimensional in a number of ways, but none of them too obvious. Three-dimensional objects can be edited with some 2D functions if viewed from the top or elevations; unfortunately, some of the editing commands I see as potentially very useful do not apply to 3D objects. The 3D viewing capabilities are rather awkward, and would require a fair degree of automation to make consistently usable for a modeling tool. These 3D limitations will require a fundamental shift in my conception of a modeling tool, but a potentially interesting one by forcing me to consider the modeler as much more *diagrammatic* than I had previously considered.

MiniCad has a built-in spreadsheet data system, which are called "worksheets" by MiniCad's publishers, that is usable in a separate window or can be displayed as a drawing/text element. Most standard spreadsheet functions seem possible and there are several special functions that return values queried to objects in the drawing database, specific to many different criteria. For instance, one can return the value of the sum of perimeters of all objects on a specific layer, or of all rectangular objects, or all of a particular shade of green, etc.

It appears as if the attribute list in MiniCad is highly customizable, and this is very encouraging. Each element can be given a unique name, be assigned to a particular "class", and have particular "records" with variable "field" values assigned to it, all user-defined. This hierarchy of data could be very useful to creating and managing object-oriented information. The MiniCad screen has a

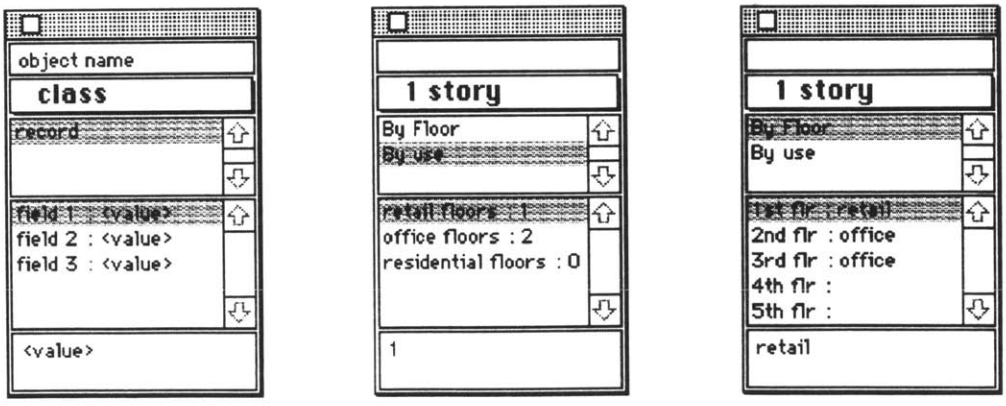


Fig. 2.3 examples of MiniCad Data Palettes

"Data Palette", visibility optional, that displays all of this specific information for an individual object when that object alone is selected. (Fig. 2.3) This is a very direct and passive form of query.

MiniCad, like most other CAD programs, had a means of creating and instantiating symbols into a drawing file. Symbols in MiniCad can take on any imaginable graphical characteristics, including 3D. Furthermore, global updating is very easy, and symbol libraries can be organized into folders and transferred easily between files.

MiniPascal has the advantage of allowing procedures to use "dialog boxes" to inform, warn, or query a user to make a particular selection or provide specific information. Dialog boxes are a standard Macintosh convention which brings a new window on top of the current working screen, disappearing once a suitable response is made. MiniPascal dialog boxes are very customizable. Another means of communicating to the user is the "message palette," a small text box that will appear and disappear in the corner of the screen which can be used to provide information or to prompt the user to perform an action.

I am most impressed with the capabilities of MiniPascal, MiniCad's programming language based on Pascal. In addition to full Pascal functionality, MiniPascal includes nearly three hundred special functions and procedures for creating, selecting, attributing, changing, querying and handling database objects in addition to calling standard menu commands. User-written commands can be maintained within a drawing file and are featured on customizable "command palettes", where a double-click will initiate the procedure and an option-key/double click will open the procedure for editing. Believe me, this ease of use has made MiniCad fully worthwhile.

A Proposal to Still Allow 3D Modeling

No longer able to pursue the "electronic clay" ideal of a solid modeling tool for the morphological design of urban areas, I looked into ways of using the two-dimensional modeler for modeling and information handling and another three-dimensional modeler for visualizing and adjusting the design. My first consideration was an attempt to export 2D proposals out of MiniCad and into

other CAD packages with full 3D functionality (perhaps "Form-Z", for instance). Once in this new program, the user could then effectively visually test and edit the model in a more "real" 3D environment. Once adjustments had been made, the model could be sent back to MiniCad where program compliance and other information management routines could be run, or be sent to yet another CAD program, with effective instantiation tools for a "greater design realization." (Table 2.1)

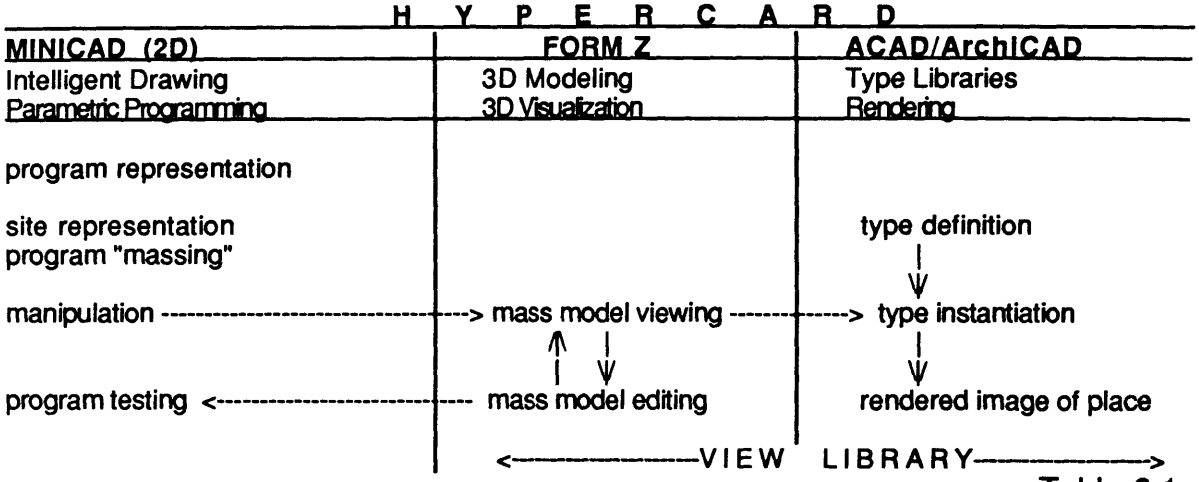


Table 2.1

This rather ambitious proposal was considered with the ideal of "portability" of data and routines in mind. This is vaguely similar to *object-oriented programming*, where a user may pick and choose special functions from a variety of applications to use in a totally personalized CAD environment. In this case, the umbrella application could be HyperCard, a Macintosh product which is very good at allowing multiple functions and applications to occur within a customized environment.

This scenario is very much beyond my capabilities, though I had hoped I might achieve at least the initial translation into Form-Z. In addition to not being able to afford the additional software, the current data translations standards (DXF, and IGES) do not allow the transfer of object data beyond the merely geometrical properties of point, line, and plane locations. Nevertheless, this digression was helpful in recognizing the potential of object-oriented programming.

Building Objects

The first elements of large-scale design projects that I considered as essentially *modelable* were the buildings themselves; the units of occupancy and function, positive upright components defining and framing the rest of the site. Within a design environment such as I imagine, the buildings are clearly *objects*, individually identifiable and definable. I will refer to the buildings as represented in this CAD program as "building/objects."

Abstractions of Building/Objects

For the purposes of schematic design, buildings are necessarily abstracted and simplified in order to make the greater design configuration comprehensible; if every function of every section of every floor of every building had to be considered this early on, the design state would be far to complex to assess and model. So essentially the building can be represented by a simple box or aggregation of boxes. (Fig. 2.4) There is no reason why the dimensions of these box forms cannot approximate the area and volume values for the buildings they represent; therefore inherent in the building/object are the gross floor areas of the real building. I felt that it would be helpful to consider buildings on a floor by floor basis, given that often the ground floor may be of a type distinct from the upper floors, and therefore having an essential impact on

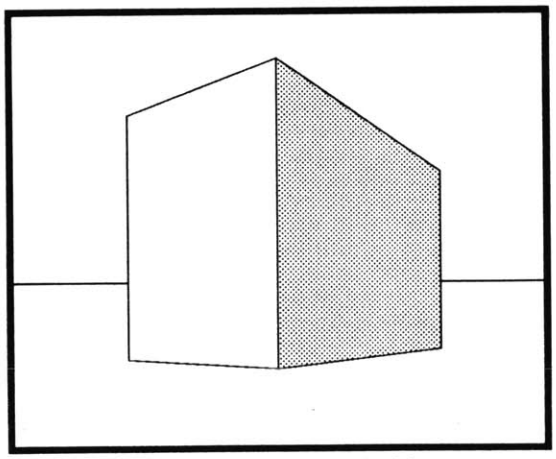


Fig. 2.4 building as abstacted object

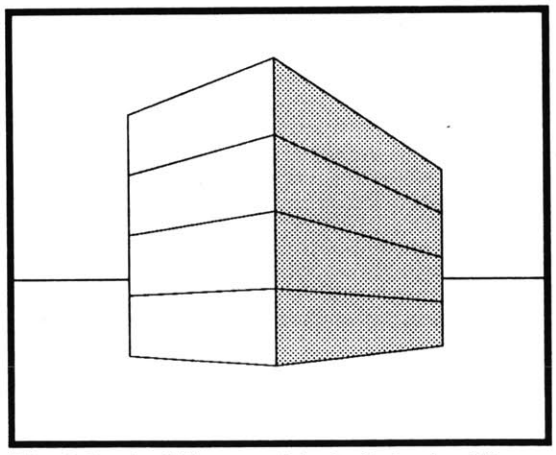


Fig. 2.5 building as object of stacked floors

the very nature of the building type. Therefore, the building/object in the schematic modeler would be a stack of boxes, each box representing a floor. (Fig. 2.5)

The buildings' general uses must also be considered during this phase of design, and a simple means of representing use would be color-coding or shading. For these purposes I have chosen (somewhat arbitrarily) red to indicate retail uses, blue to indicate office, and yellow to indicate residential uses. (For illustration purposes, consider three shades of gray. Fig. 2.6) This information too becomes inherent and essential to the object and like the box dimensions constitutes the data of the building/object.

Conventions of Two-Dimensional Representation

I have already spoken of the 3D limitations of MiniCad, and these originally considered conventions are simply not usable in a schematic modeler on this platform. I therefore had to consider some means of representing this same information in a 2D object, hopefully allowing the user to recognize the 3D properties of the building/object and consider them accordingly. Fortunately MiniCad and MiniPascal have several object features that are useful for conveying "extra-dimensional" information (i.e., beyond merely width and depth). There is a special polygon feature that will only consider rectangles, flat

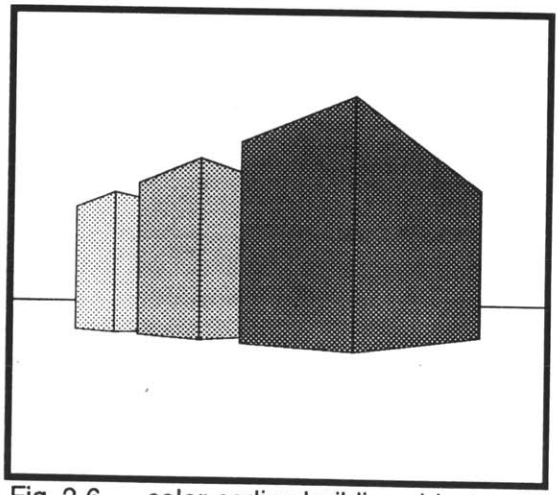


Fig. 2.6 color-coding building objects

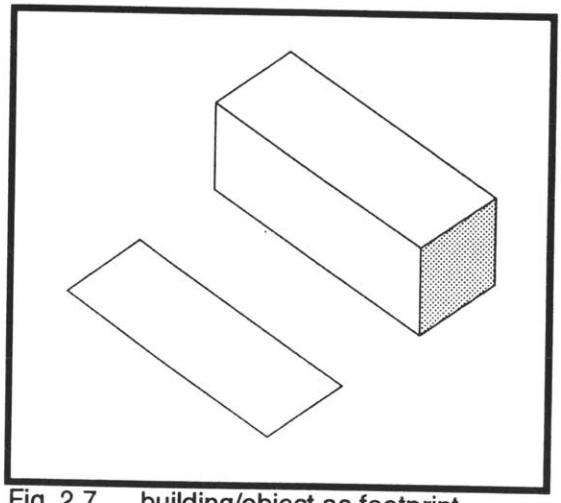


Fig. 2.7 building/object as footprint

four-sided objects with right angles and parallel sides; this is in keeping with fundamental properties of building footprints. (Fig. 2.7)

These rectangle objects delimit finite planes that can be filled with colors and patterns. The color conventions may remain as originally considered. However, when considering a 2D plan, which is essentially a top view, only the top floor is visible, and the user lacks any indication of what other uses are included in the building/object. MiniCad has customizable hatch patterns comprised of foreground and background colors, also customizable. The mixed uses in a building/object may have their representative colors included as the foreground and background colors of the hatch pattern. Additionally, the relative amount of one color to the other in the hatch may roughly represent the proportion of the one use to the other. I have chosen a series of simple diagonal hatch patterns for this purpose. (Fig. 2.8)

Finally, and perhaps most importantly, some convention must be used to represent the height of the building/object. There are several conventions already for representing depth on a conventional plan drawing, notably isometric and shadowing techniques. Variations of both were considered for this modeler but proved too difficult to maintain as the objects are edited. I've had to settle for the one remaining graphic attribute of polygons: the line weight of the polygon's edge. Though not terribly attractive or convincing as 3D, the added "weight" to the object reads somewhat intuitively as a higher density object. (Fig. 2.9)

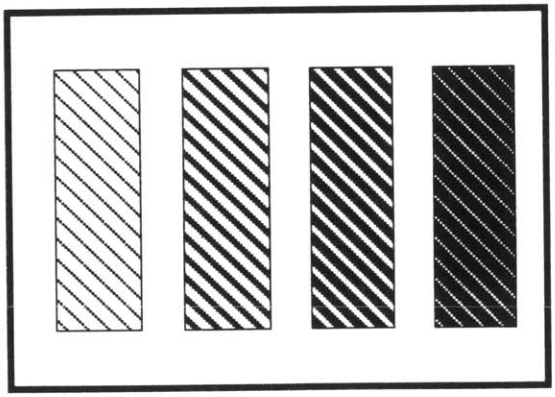


Fig. 2.8 proportion-indicating hatches

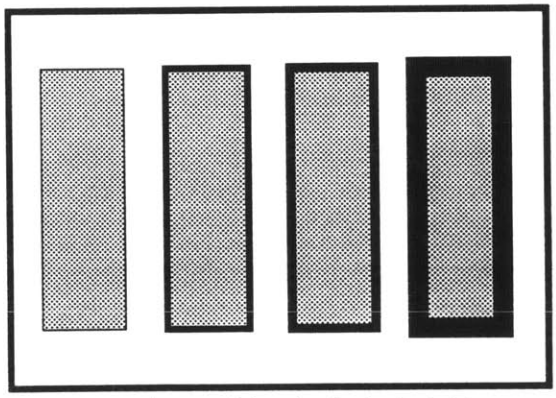


Fig. 2.9 line weights indicate height

Essentially, the representations I have proposed for building/objects in lieu of an appropriate 3D modeler are *diagrams*, and as such carry a significant responsibility to represent effectively their intended ideas. There has been much research addressing the power and functions of diagrams to design (Arnheim 1969, Ervin, Fargas 1991), but the conventions I have developed here are contrived from my own common sense to solve practical problems and limitations. The assumption is that the user of this environment would be adaptable to understanding the objects' appearances and their associated data characteristics.

Building/Object Data

Graphic attributes alone are not entirely reliable for holding the data of the building/objects. For instance, the mixed-use ratio indicator as represented by the hatch pattern is only an approximation. The user and program should not have to rely on this information alone to assess the building/object. Fortunately, MiniCad's customizable class and record/field data attributes and Data Palette have proven very useful in this regard.

Objects in MiniCad may be assigned to classes and assigned records independently. Each record may have any number of fields which may in turn hold any value (of a specified type, i.e. integer, real number, or string). Any one of these data may be specified as a criteria when performing computations or actions in the program. Though I do not really understand MiniCad's intention in providing this particular organization of object data ("name," "class," and "records" with variable "field" values) I found it somewhat flexible, and it suited my purposes well enough.

I decided to set up two sets or record/fields for maintaining data pertaining to the building/object's uses. The first record indicates use "By Floor", with fields representing "1st Floor", "2nd Floor", "3rd Floor", etc. (up to five floors for now). The field values are strings signifying what use is designated to that floor: "retail", "office", or "residential." Fields for floors higher than the building/object is tall are left blank. Alternately, I have a second record "By Use" which has as values integers indicating the number of floors each use has designated in the

building/object: for instance, field "office floors:" might have a value of "3" indicating that three of the floors of the building/object have been designated for office use. These different record configurations are redundant, but I intended to support both to allow greater and more efficient querying of objects in subsequent editing and tabulation routines. For object class, I have again been redundant for the same reasons; defined classes would be "1 story", "2 stories", "3 stories", etc., of course indicating building height. This textual data assigning serves to more clearly inform the user of the characteristics of the building/ object by displaying all this information on the Data Palette, standard to MiniCad. A query of an individual building/object would therefore be as simple as selecting that object with the cursor and reading the Data Palette. (Fig. 2.3)

Editing Building/Objects

Design Moves

When it comes to editing or acting upon the elements in a design, the assumptions I have made become more pronounced and debatable. I had intended to provide what was essentially a sort of "electronic" clay, to allow the designer to do 3D massing and morphological studies on the computer much as s/he would with this traditional modeling material, but with the added benefit of imbuing knowledge into the model elements. By modeling in the traditional way, the designer would be free to address and explore the relationship of forms and voids in three dimensions without the undue burden of material and functional limitations. My goal was to simulate this experience on the computer screen, while also allowing the designer to engage in a further dialog with the design state with regard to quantitative issues.

Of course the limitations imposed by MiniCad will not allow this sort of modeling environment, so I was forced to reconsider. Modeling in two-dimensions could be considered similar to working with paper cut-outs, much as designers develop compositions in collage; this is a poor substitute for clay for exploring a three-dimensional composition, but with the line weight, color, and hatch conventions proposed, progress is possible. These visual clues are modest improvements on the paper-based equivalent but significant. Add to this the

use of object-bound information then we now have the potential for a fairly powerful modeling tool.

While many design moves are as simple as changing the position or location of an object, others imply more complex transformations. This is especially true with the building/objects as defined previously; to stack a building/object on top of another is to create an entirely new building/object, with a different class, a different set of field values, and entirely different graphic attributes. For the more basic moves, standard MiniCad commands for moving and rotating objects will do. For others, such as stacking or clipping objects (changing their shape), special procedures would need to be written to account for and maintain consistent object-oriented data and graphic characteristics.

Building/Object Creation

One critical point is the issue of building/object creation for the design model. This raises the questions of initial design moves (based upon "what is known" before designing actually begins), and of *what reason* the building/object has to exist: to satisfy a development or planning program requirement. Some authority or power, whether individual or public policy or market demand, has determined that some amount of built space of certain use is desirable for a site. This in itself is information useful to the designer and to the design tools s/he uses. Therefore I have included a means of incorporating this information into the design environment through input of the building program square footages, to be used at the very start of a design project.

The program input and building/object creation routines were the first serious MiniPascal procedures I wrote for this project. They are necessarily simplistic, allowing for limited interpretation of the development requirements; I did not want to spend a disproportionate amount of time on just this initial step. Nevertheless, I did provide two different means of input: one, called "Prgrm Areas", requests the total required gross square footages by use (retail, office, and residential), and another, called "Prgrm Pcts.", requests the total combined project square footage and the percent of the total designated for each use. These requests and their input responses are handled with customized dialog boxes.

These procedures then create a set of building/objects for this total required project area. The objects are single-story units and have automatically determined graphic and data attributes according to the conventions already outlined. Furthermore, the objects come in predetermined standard widths typical to their use (for instance, 60' wide for residential and office building/objects and 35' for residential). These constants can be changed by editing the procedures manually, although they could be changed automatically via dialog boxes, if more time were available. These building/objects are lined up in a reserved section of the design work area in lengths appropriate to the page setup. (Fig. 2.10) These objects now constitute the "material" for the design, having preset information content that subsequent routines and the designer may utilize as the design database.

This set-up brings us to a brief consideration of the graphic format of this specialized MiniCad work environment. The area where the design modeling would occur should be as large as possible to allow the maximum scale representation; this area I will refer to as the "design space." The reserved area just mentioned will remain in active use during the design process, handling residual, or leftover, areas as needed; this will therefore be referred to as the

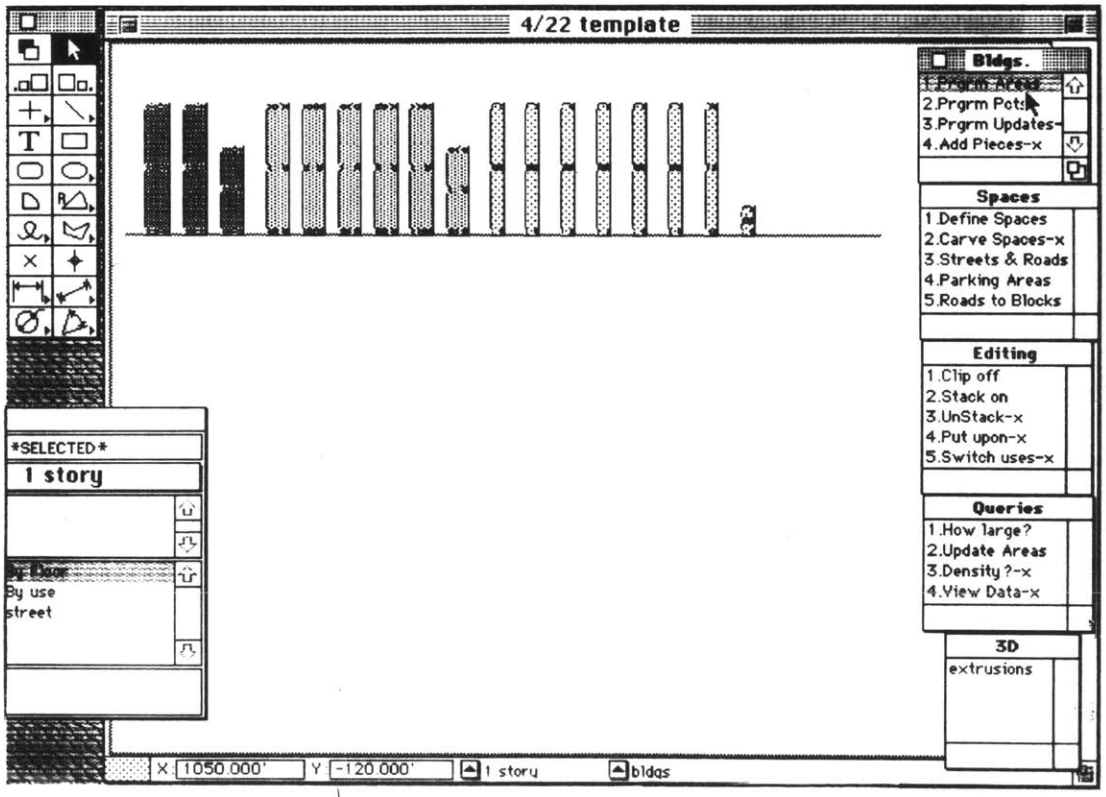
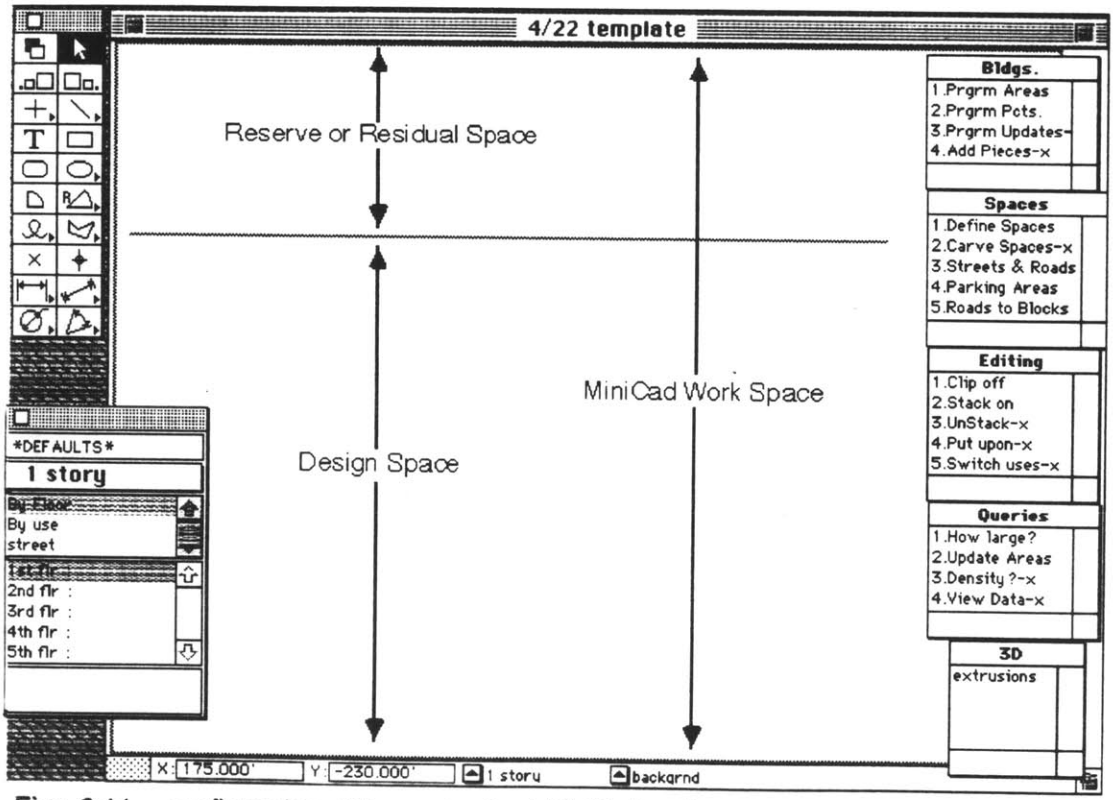


Fig. 2.10 building/objects in reserve space as input by "Prgrm Areas"

"reserve space" or "residual space". The command palettes unfortunately cannot be hidden and recovered very easily and must remain on screen. I will be referring to the entire screen as the "work space." The arrangement I have settled on for the work space is shown in Figure 2.11.

Sizing and Positioning of Building/Objects

The user may then begin modeling with this "material", bringing the objects down from the reserve space for incorporation into the design space. The project site is assumed to already occupy the design space , whether pre-drawn in MiniCad or imported from another application. Of course, the building/objects in their current form are not suitable for much designing and will require acting upon to make them the right sizes and orientations to suit the designer's intentions. As I said, some of the simpler actions, such as moving and rotating, can be accomplished with standard MiniCad menu commands, since they require no fundamental change in the building/object's information, that is in its size, use, etc. (By the way, MiniCad is rather poor at allowing non-rectilinear or rotated-grid designing with special rectangular polygons. For purposes of simplification, all designs for the time being are assumed to align to a single vertical/horizontal orientation.)



Figs. 2.11 configuration of the customized MiniCad work space

The first method I considered and implemented for size-adjusting of building/objects was a clipping tool that, much like scissors on paper, clips the rectangles along designated edges, offsetting the new object clippings from the original pieces. (Fig. 2.12) "Clip off" is accomplished while maintaining consistent total area and object information; clipping a complex stacked building/object maintains the same complex attributes in all newly created pieces. This was no easy task in MiniPascal, and required a complex series of operations to maintain the original data and to transfer them to the new objects (see Appendix 2).

This emphasis on creating new objects while maintaining consistent areas was based on the assumptions that this was the best means of managing the total design area - neither creating nor deleting new areas. But upon further study, this approach creates more problems than it solves. Consider, for instance, if the user has achieved a design solution that s/he would like to alter a bit, say, extending a building by ten feet. Must the user then go to the reserve area, clip off a ten foot section, return to the building in question and abut the new piece to the building? Not only is this a lot of trouble, but the combined pieces have no consistency of data. The clipping tool is still useful for creating smaller building/objects out of the large initial objects, but must not be relied on for all sizing operations.

For this reason, resizing of individual building/objects must be allowed even if it means changing the area of the object. This will greatly facilitate the modeling

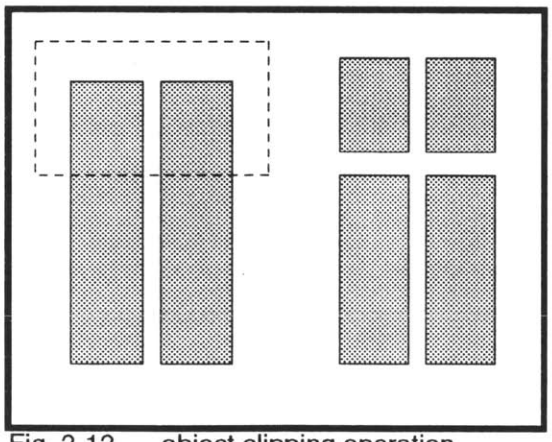


Fig. 2.12 object clipping operation, before and after

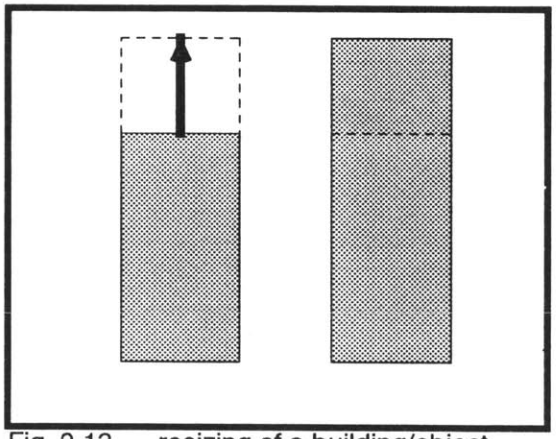


Fig. 2.13 resizing of a building/object, before and after

process, and the maintenance of the initially programmed areas may be handled with specific routines for information management (using areas still in the reserve or residual space; details to be covered later). The resizing tool is another standard MIniCad function where the user simply uses the mouse to pull and drag the edge of an object the desired distance. (Fig. 2.13)

Stacking Building/Objects

A more complex transformation of a building/object is to extend it into the third dimension. The original pieces created after the input of the development program are only one story objects, and a means must be provided for creating more complex building/objects representing multi-stories from these single story pieces, automatically adjusting the graphic and data attributes to conform to the user's intentions. The command written for this purpose, called "Stack On," first prompts the user to select with the mouse the objects to combine, selecting the "object to stack upon" first, and then selecting the "object to stack onto the previously selected object." (Fig. 2.14) The order and wording of these prompts are important to make the user consider which object is to serve as the "base" for the stack. This consideration is important because the order of uses from the ground level upward is critical to a building's identity and function, and the new building/object must reflect this intended order. Furthermore, the "object to stack upon" as base will provide the footprint for the new multi-story building/ object, maintaining a consistent size and position for the new object; in a sense, a one-story building in a plan may be converted to a two or more story building by stacking other objects upon it.

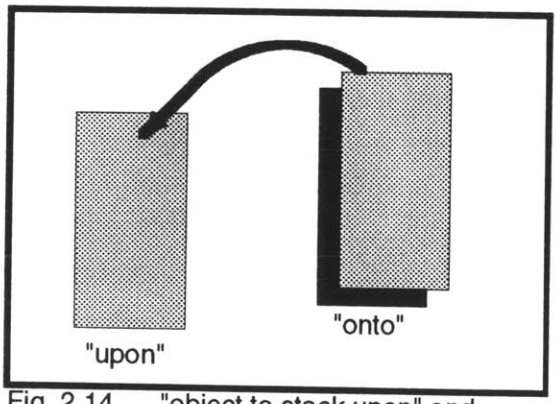


Fig. 2.14 "object to stack upon" and "object to stack onto"

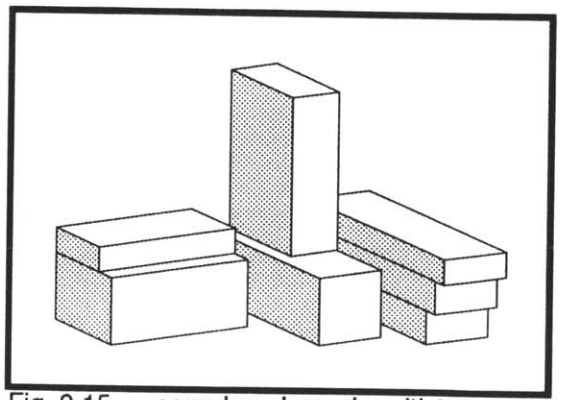


Fig. 2.15 complex-shaped multistory building/objects

This operation involves other issues regarding maintenance of a consistent total area of objects. The first is that when one object is stacked upon another, the first object no longer occupies an area in the design space; it is "virtually" on top of the old one, and the original appearance of that object has been deleted. The new stacked object must now represent an area twice what it actually occupies on screen. This is where the data attributes begin to serve their purpose. If queried, this new stacked object will assert itself as a doubled area by using the data it has of itself of being two stories as a factor in any area calculations. Therefore, the total calculated area in the design space would remain constant.

However, we must consider the other issue that arises concerning area consistency when stacking. Suppose the object to stack onto is not of the same area as the object to stack upon? If it is larger or smaller, then the "virtual" area of the new object, double the object to stack upon, would be inaccurate. This discrepancy will be handled the same way that the "resize" option mentioned earlier is handled, with specific information handling routines to maintain a consistent total programmed area.

A variation of this stacking tool, but not realized yet due to time constraints, would be a routine for "unstacking" a complex building/object. The usefulness is obvious, considering that it is the designer's utmost prerogative to change her mind. I am not sure if this procedure should completely deconstruct the complex object, making five individual single story objects of a five story complex object, or if it should unstack objects a floor at a time (perhaps a choice between these options would be nice). I have also not decided if the unstacked objects would be accounted for as "residuals" as mentioned for resizing and for stacking unequal objects. More likely, the individual components would maintain their size and proportion but be arrayed in a separate area, to allow the option of restacking if desired.

Another variation of the stacking tool, also not realized yet, will allow for the incorporation of more complex-shaped multi-storied objects, permitting setbacks and overhangs if desired. (Fig. 2.15) The strategy here would be not of stacking, but of "putting upon" building/objects on top of other building/objects, but not integrating them as single complex objects. Area

consistency would be maintained, but several issues remain unresolved and deserve further study. For instance, could this new complexly shaped building/object be effectively represented in 2D? What data attribute would hold the relevant information regarding this new type of position? These questions would become more acute when requiring true 3D realization of the building. For now, the design environment will not consider such complexly stacked buildings.

Information Handler Component

Information in the MiniCad Design Environment

By imbuing the objects of the design with information a database is created which allows much greater manipulation and "feedback" on the design status by procedures than with simple, unknowledgable objects. Unlike the traditional computer database of tables of values, this database is "object-oriented," allowing more direct manipulation of the data for constructive purposes; manipulating the objects of the model is the same as manipulating the design database. There is no need to have special features tracking each object in the design, where it is, how big it is, and its other data features. The data is not charted; it just *is*, much as it is in the world around us.

My general strategy for handling this information is simply to make it accessible to the designer when s/he decides it is needed, in a form that is useful or meaningful. There are many different degrees by which information can assert itself for constraining purposes, from strictly not allowing some moves, to suggesting alternatives, to simple advising that a preset threshold has been reached. The more aggressive ones I admit are beyond my ability, and perhaps MIniCad's ability, to implement, requiring numerous object-inherent rules and continual evaluation. Relying on the user's best judgement to request and assess the information constitutes a very passive form of constraint, and is the approach I advocate and will use.

MiniCad Worksheets

MiniCad's built-in spreadsheets, or "worksheets" as they are called, would be useful for compiling the information of the object database into totals, subtotals, ratios, whatever the user might deem important to the design situation, and for presenting that information in a logical form. The row and column format works much like any other spreadsheet program, most of them variations on Lotus' very popular program "1-2-3." Having worked with spreadsheets on several occasions previously, I was fully prepared to take total advantage of this capability to convey the design's information in significant ways. Unfortunately, the developers of MiniCad are guilty of an oversight. Simply put, of the nearly 300 special functions in MiniPascal, there is no function which can send operations to a worksheet that is not pre-selected, and in a "Catch-22", no commands may be selected without de-selecting all worksheets. This of course severely limits automation of worksheet calculations. Worksheets may still be updated and recalculated by manually selecting a recalculation command, but this limited utility (limited in relation to my hopes) was enough to discourage me from utilizing worksheets as information-handling tools. I was interested in exploring what automation I could implement, and this would be merely using what MiniCad had already developed.

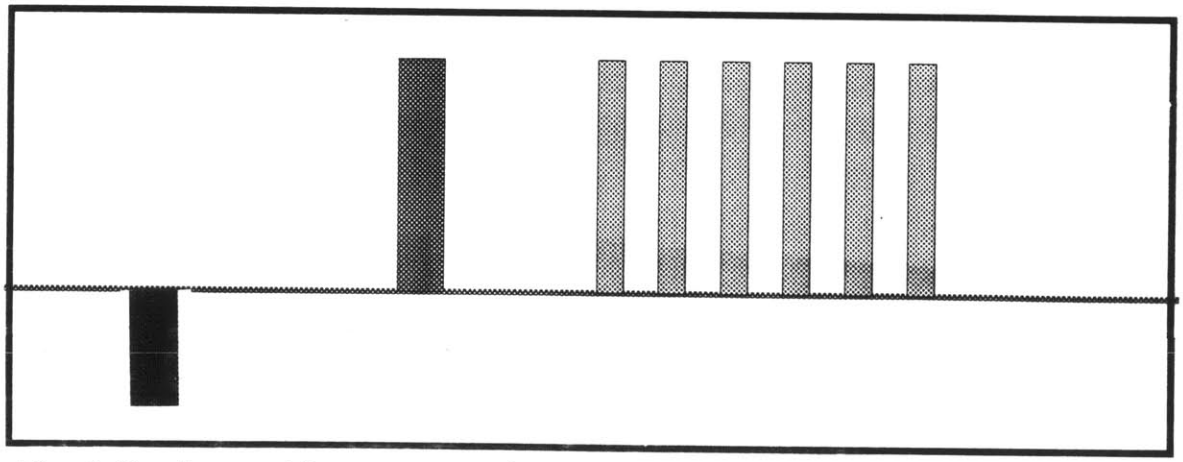
The greatest handicap that this worksheet inaccessibility caused was the difficulty in maintaining global values, that is, values that are to remain unchanged in the design environment. Each object may have a value, and their sums are be values, but these are never constant. There seemed no way to maintain the constant values representing the building program which was input at the initiation of the design. I have referred to some editing commands which allow the total area to deviate from the original value. To rectify this situation, I needed a command which would adjust the objects in the residual space to again meet the preset values, and for this I needed this command to refer to those values.

An Alternative Approach to Area Monitoring

The solution I used was to create one special "inert" object, not a part of the design and unchanging, which would hold these input values in record/fields. I

considered making this special object totally arbitrary, obscure in some corner of the screen, but instead I found a particular, unexpected use for it as the "baseline" to the reserve/residual area of the screen. (Fig. 2.11) With the data safely stored in this screen-formatting element, any commands I would write could refer to this object and its data as a globally constant value in the design space. Furthermore, these values could be acted upon and updated as the development program itself is updated, a necessary flexibility considering that often the design goal changes as the design itself evolves. (Note: This program updating routine has not yet been implemented.)

The method I used for maintaining the programmed values and informing the designer as to conformance of the design to these values was rather unexpected. I originally thought a tabular display of numerical values, indicating the set value, the design value, and the variance would be appropriate. But given the problems with using worksheets and the already implemented reserve/residual feature in the work space, I saw the opportunity to make this feedback more visual and graphic for the designer. I wrote a command which, when selected by the user, would update the objects in the reserve space to indicate the balance left over between the objects in the design space and the preset values, essentially keeping the total gross areas of the entire work space constant. This routine involves summing all the building/objects in the design space, retrieving the initial values stored in the inert "baseline", calculating the difference, and replacing the remaining building/objects in the reserve space with a new set. If the design has used



Figs. 2.16 "bar-graph" type representation of reserve space status; negative value for retail space shown

more than its allotted area, then the reserve can indicate a negative area, represented as a building/object *below* the baseline of the reserve space; this begins to resemble a bar graph and is easily comprehended. (Fig. 2.16) Such a precise update and its visual feedback all but eliminates the need for a threshold warning.

I considered the possibility of making this automated update of areas a continual process, correcting the reserve set whenever a change occurs in the design space. The problem with this is essentially one of programming and of processor power. The procedure for updating takes every object in the work space into account, and this is time-consuming for the computer processor; I would be afraid the pause between commands would be distracting for a designer. Also, the procedure for updating is rather lengthy and therefore not "portable" enough to include in every command that involves changing objects; the time taken to read and compile the commands would increase as a result.

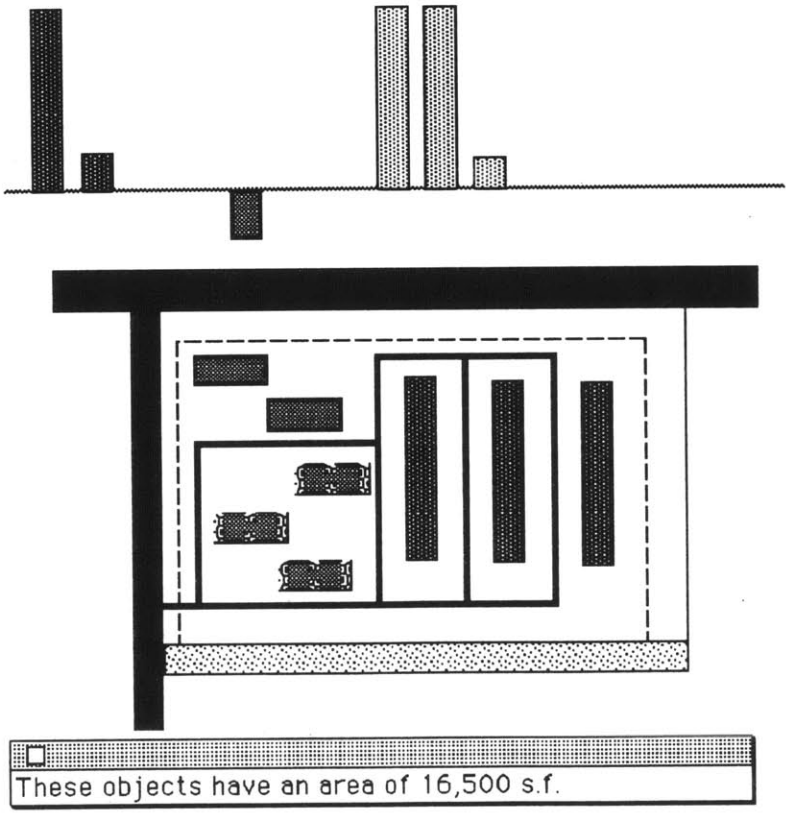


Fig. 2.17 message palette displaying results of "Area?" query command

Furthermore, standard MiniCad commands for editing would not have access to the updating procedure, leading eventually to inconsistent application of it. There *may* be some form of allowance for on-going or "transparent" procedures that activate when some special event occurs in the work space. This would certainly be more efficient and consistent, but time constraints have not allowed me to pursue this option. As it is, the update tool allows flexibility and user discretion, qualities of this modeling environment that I consider beneficial to design activity.

Special Queries

There remains the issue of specialized queries to the design, such as what is the total floor area of a particular building/object or of a group of building/objects, say a block or district. These operations are relatively simple; the user first selects the object or objects to ask this question of and then selects the particular command. The total is displayed in the message palette. (Fig. 2.17) So far the only such query procedure I have written is for calculating total gross floor area of the selected objects, called "Area?". It would be relatively simple to write similar such commands for other types of queries. Another might be: What is the total retail (or office, or residential) area within the selected objects?

Ideally, the queries would be flexible, allowing the user to inquire about any sort of quantitative characteristics of the design database. The number of possible queries is very large, and some could become quite complex. A user may want to apply a query only to certain sections of the design area, such as the density or F.A.R. of a particular block. Using tools such as a structured query language to set up and investigate complex relationships would be very helpful. I can imagine that such procedures might be possible, perhaps on a more sophisticated platform, using a dialog box to specify the desired query. Once again, time has not allowed me to explore this option very far.

Using Roads in the Modeler

Of course, there are several other elements involved in large-scale design besides building masses. Although in my investigation I did not consider road elements until rather late, I am presenting the issue now because I consider them rather fundamental to the early evolution of a design project. The road network of a new development usually parallels the infrastructure of the project, and as such is often the framework or organizing principle for the entire design. The design and definition of the road network is therefore a critical step, and one that by all means must be included in the schematic modeling tool.

First of all, I considered what characteristics of roads were significant and how those characteristics may be represented or maintained as data in road/objects. Their primary function is normally one of access throughout the site, providing connections to places and to one another, the latter occurring at intersections. They are without exception linear, and have varying widths usually correlating to anticipated traffic volume. Some may be divided as boulevards.

As objects in a 2D schematic modeler, roads may be indicated as line elements in recognition of their fundamental connectiveness and linearity. These connections from intersection to intersection, or block segments, often have characteristics of their own independent of the entire length of the road, and so may be defined separately. The significance of the road with respect to traffic function, that is traffic volume, should be defined by the number of lanes and indicated by line weight. We may also consider on-street parking a characteristic of many roads, having an impact on the road width and also on the programmatic function of the entire project.

Therefore, the command for creating road/objects, called "Roads," prompts the user for the number of lanes of the next segment to create as well as for the number of sides of on-street parking (0, 1 or 2). The user is then prompted to pick begin- and end-points in the design space, and is then asked if s/he would like to continue creating road segments, freeing the user from repeatedly issuing the command. The former values for lanes and parking are maintained as defaults which the user may change if necessary. The "smart cursor" standard in MiniCad insures that clean intersections are made. The finished

road/object is automatically represented by a line that has a weight proportional to the number of lanes defined for it, and the object's data includes the number of lanes, the on-street parking condition, and a value for the total road width (assuming and average of 10' per lane of road and parking). (Fig. 2.18) These values are held in a special record/field specifically for road/objects. Editing is as simple as picking a road segment and moving it or its end points to a desired location. There ought to be a means for redefining the lane and parking values, as well as for indicating boulevarded roads, but these have not yet been implemented.

The resulting network of line objects is a very schematic representation of the designed system of infrastructure and conveyance, but may be too schematic for many designers. The problem with line weights on a computer display is that they are independent of the scale of the current view, and remain a constant number of pixels wide no matter how much one zooms in. As such, they are not really useful for accurate dimensional positioning of buildings, particularly if sidewalk width is a prevailing concern. For this reason I have looked into alternative representations of this element. "Real Roads" creates from the line segments rectangular polygons of the width indicated in the record/field. (Fig. 2.19) Using this tool is left to the user's discretion. It serves the purpose of transforming only certain schematic ideas and representations towards a greater realization; this realization is limited so that building/object and space/object modeling may continue at a schematic level. The drawback of this representation of roads is that it is not easily edited without getting deeply involved in coordinating numerous vertices (the corners of the polygons). Therefore, the original road/object line network is maintained after the

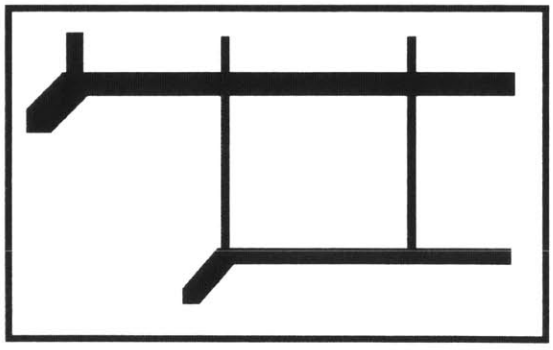


Fig. 2.18 road/objects in the design space: lines with weights

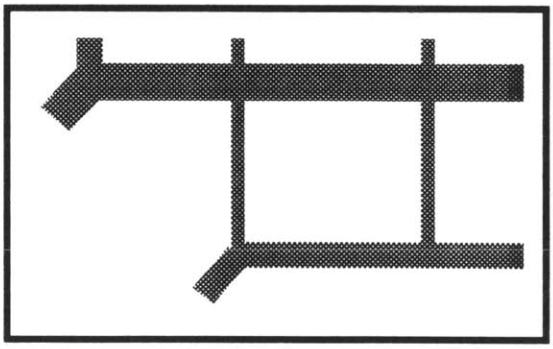


Fig. 2.19 realized road/objects - real width dimensions

procedure so that if editing is required, the user may go back to this level and make changes, reinvoking the "Real Roads" command to return to the same level of realization.

Like building/objects, road/objects also carry inherent data available for querying, although these quantities are probably of less critical concern. Issues such as amount of paving may be determined. Of more significance would be the amount of on-street parking available. The best use that I have for such information has to do with the greater design realization component to be discussed later.

Space/Objects

Another very important element in large-scale designs is of course the public, outdoor space; in a sense, spaces are probably the most critical for achieving a pleasant design solution. Exactly how to consider spaces, what they are, what they accomplish, what makes for good ones, are issues upon which I will probably make my most bold assumptions. I don't expect my approach to satisfy everyone. Even considering spaces as objects themselves involves presumptions: what are considered the dimensions of this object? what are the pieces involved? how is it defined?

What are Spaces? (my approach)

I consider that since spaces are quite literally voids, or non-things, it is the objects around them that delimit and define their qualities. If bounded by office buildings. the space would of course be different than if bounded by residences, or by festive retail. The spaces in turn affect the surrounding objects, serving as a basis of formal order and relating the various building facades to one another over distance. In a sense, I consider that the building facades *belong to* the space they front; as will be demonstrated later, this idea will serve as a foundation for the greater design realization tool.

The ground surface of the space has characteristics of its own, such as paved or grass surface, pathways, plantings, seating, and focal points (on the ground, but

influencing spatial character and dynamics), which in my view serve to arbitrate between the surrounding objects. There are other socially-oriented characteristics of spaces which are also important, but considering that this is strictly a morphological design tool, these issues are left to the designer to handle.

Modeling with Spaces

The purpose of space/objects to the modeler is to give order and sense to the areas between building/objects. One common method of considering the spatial structure of a district is a figure/ground reversal, where the spaces become positive objects in the composition, discrete "rooms" of individual character and experience. (Fig. 2.20) Including space/objects in the 2D schematic modeler attempts to cater to this method. By constructing a system of spaces, and working with the building/objects to fit into that structure, and alternative design state is effectively allowed by the modeler. On the computer environment, other layers may be set to "invisible" or "grayed" to make the spatial structure the positive element in the design space. For simplicity's sake, I assume that *individual* space/objects are discrete and have geometric integrity, being not concave nor overly complex in shape.

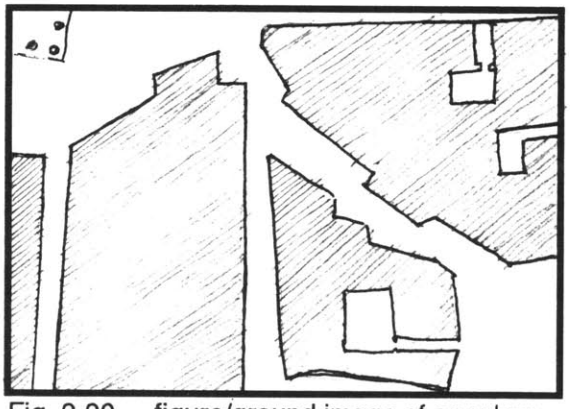


Fig. 2.20 figure/ground image of an urban district (from Nolli)

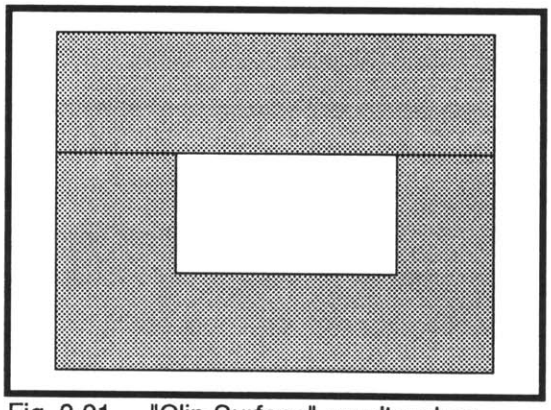


Fig. 2.21 "Clip Surface" results when clipping a hole in a larger surface

Creating Space/Objects

I initially considered making the geometric distinctions of the spaces in the modeler automatic, relying on a sort of positive/negative switching procedure which would determine integral pieces of the area of the design space not already occupied by a building/object. One of MiniCad's standard functions is Clip Surface, which removes from an object below the area occupied by the object above. Clip Surface cannot leave simple holes in an object; rather it must split the larger object into two objects, one concave where the area has been clipped out. (Fig. 2.21) I had hoped that using this command for all the

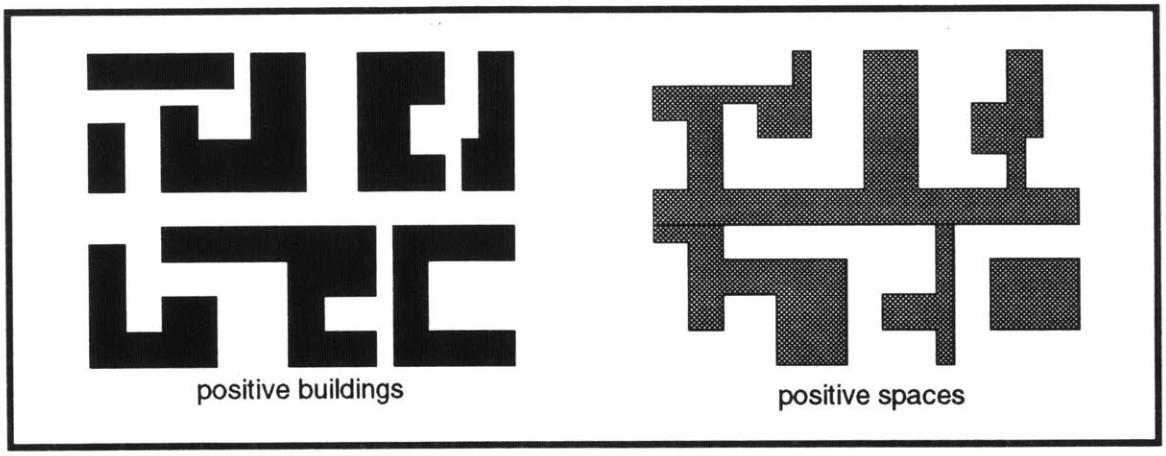


Fig. 2.22 poor results of an automated positive/negative method of space-defining

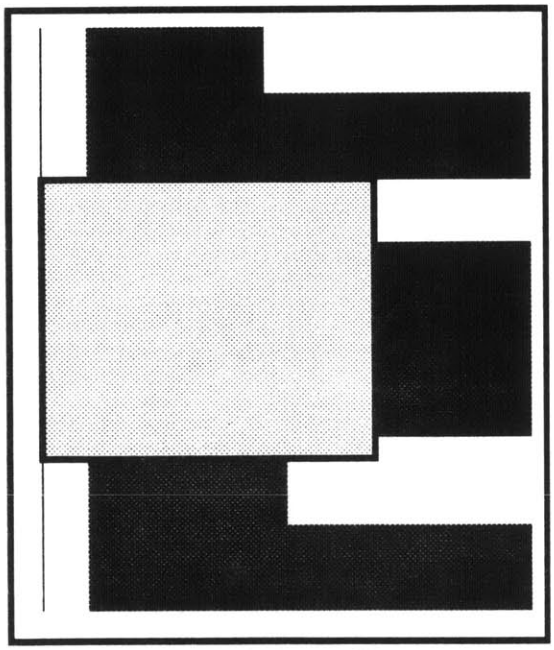


Fig. 2.23 space delineated between building-objects

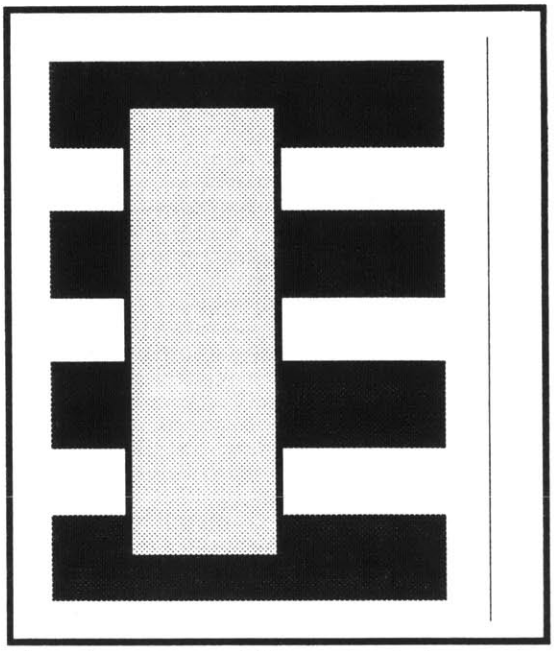


Fig. 2.24 space delineated out of building-objects

building/objects on a surface representing the site might create objects that resembled discrete spaces. Unfortunately, the rules that Clip Surface uses are not geared towards the geometric integrity of the resultant objects, leaving unusable pieces instead. (Fig. 2.22)

Instead, I have created two commands for defining the edges of spaces in 2D. The first method, for delineating the spaces between building/objects, calls for the user to draw a polygon around the edges of the intended space, perhaps, but not necessarily, using the surrounding objects as guides. (Fig. 2.23) (Note: Since the designs so far can only be rectilinear, I could have used the draw rectangle function of the program. However, for purposes of trying different routines I opted to allow drawing of polygons of any shape.) When the begin point of the polygon is selected a second time, the user is given a fill tool to pick the interior of the space (this part is not quite automatable). There is not yet any sort of error trapping implemented to check for overlaps with other objects; the user must use his or her better judgement in this case. The second method utilizes part of the procedure written for clipping building/objects, and is used here instead to "clip out" areas from existing building/object configurations; a sort of "carved space" approach. (Fig. 2.24)

Defining Space/Objects

There still remains the issue of what are the specific characteristics of spaces which may be and should be defined as data to space/objects in the MiniCad design environment. The most primary distinction to make is what *type* of space is the object to represent. With this knowledge, a whole set of assumptions would follow which could be programmed in the realizer as *rules* (and covered in more detail in that section). For reasons of simplification, I considered only three primary types: plazas, parks, and streets. Two spaces of the same primary type may differ in their proportions, their context (surrounding objects), and in specifically defined *qualities* intended for them, i.e. a designer may wish a space to be *formal* or *informal*, regardless of its type or context. The designer may desire to specify arcades on some edges or specific focal points within the space. These additional elements could be specified for any space, held as sub-objects in a MiniCad "group" with the space/object, or not specified; this would be the user's decision.

After delineating a space/object, the user is queried to supply information about the type of space the object is to be and whether it is formal or informal in composition. Once again, this information is stored in class and record/field attributes. Graphically, space/objects have a paler hue so as not to detract from the usually positive building/objects. Though I have not yet provided any graphic indicator as to type (although like building/objects, color variations would do well), I have used different hatch patterns to indicate whether the space is to be formal or informal. (Fig. 2.26) The query would also permit the user to designate edges for arcades and points of focus within a space if desired. (Not yet implemented) An arcade, represented by a line symbol (grayed and thick) and a focus, a small square symbol, would become part of the object "group" of the designated space/object, and would be referred to later for greater design realization.

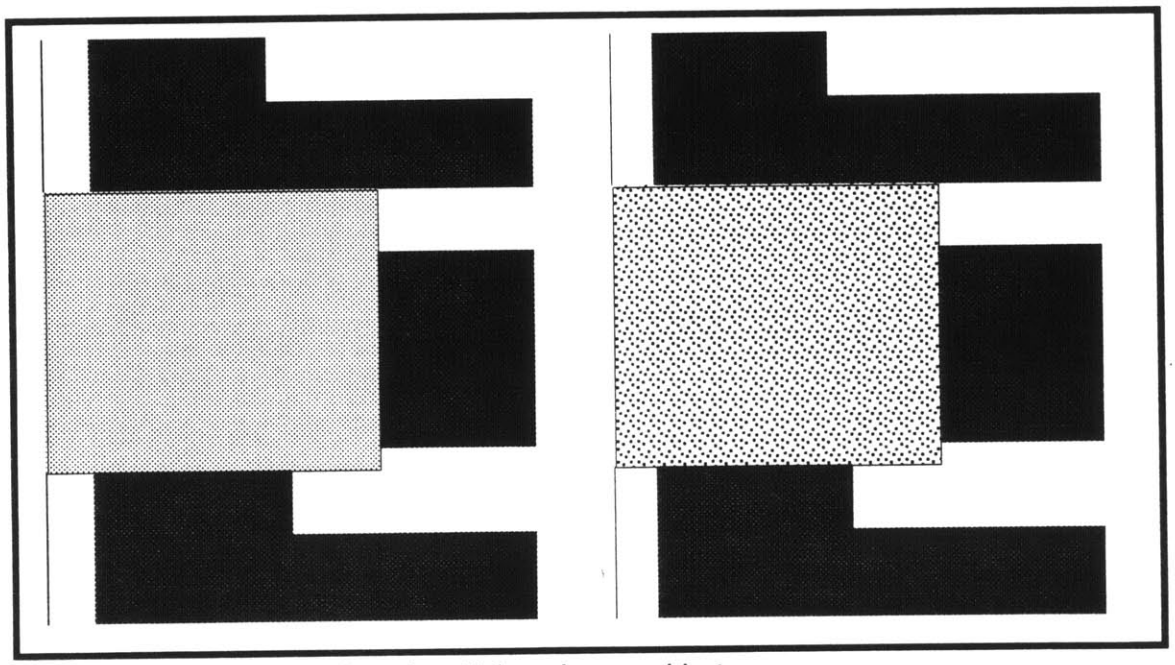


Fig. 2.26 fill patterns for formal and informal space objects

The Site as Object

I had overlooked this final element to designing of large-scale built environments as itself a potential information-bound object until quite late in the project. As such, nothing has been done by way of implementing any ideas regarding the site/object. But the ideas are valid and interesting to the notion of designing with intelligent objects.

On several occasions I had considered the importance of the site and its particular characteristics to the modeling tool but had not found any direct correlation to the concepts I had been exploring. I had settled upon simply using the site as an inert backdrop to the design space, upon which the designer was to bring his or her full knowledge to bear. Making that extra step of considering the site as yet one more type of object now seems obvious.

A site for a large-scale development is clearly a much more complex object than an individual building, and its characteristics could not so easily be imbued in a single polygon. Perhaps if the object itself were complex, say as a "group" such as MiniCad allows (aggregations of many dissimilar objects into a single entity), then appropriate knowledge could then be object-bound.

A site has boundaries, obviously, as determined by ownership or government or community incorporation (individual lot boundaries may be included as sub-group polygons or lines). This could be a basic polygon following the site edges, and would be the foundation for the object grouping. I would suggest a very neutral color, perhaps an off-white, to serve as a blank field for the design yet discernable from the white of the work space. Geometric qualities such a perimeter or area are inherent in the object, but other factors such as latitude and longitude, altitude, and cardinal orientation could be saved in fields and later serve functions such as seasonal shadow studies or climate/wind studies.

Other site properties include setbacks and designated unbuildable areas, such as wetlands or woods. These in a sense are sub-boundaries, and as polygons themselves could have functions that would not allow other objects to be placed on them (a passive constrainer that would only alert when a checking command is called). Again, area and perimeter are geometrically inherent, though values for percentage or ratio of unbuildable to buildable may be held as data values

in fields. There could also be a designation of what type of unbuildable area it is, which would also call for different graphic representations (color or hatch pattern).

Sites also have as properties government-mandated characteristics such as floor-area ratios and height restrictions. These properties apply to buildable areas only, of course, and often are have different values for designated zones on the site. Again, polygons are an appropriate object type, but this time I would suggest leaving them unfilled, or transparent, and having their outlines dashed. These graphic properties maintain a unity to the whole site and do not visually enforce the zones too strongly. The specific F.A.R. and height values may be stored as data in record/fields for each polygon. Like the unbuildable areas, these values could have a constraining effect only when called upon with a specific command. I would expect these procedures to be especially complex, but probably doable within MiniPascal. An interesting possibility presents itself with these objects: given polygons with specific areas and with floor area ratios, a maximum allowable built area can be determined. This new value can in some way be reflected in the set of building/objects in the reserve space, perhaps aiding in capacity analyses for developers. The ramifications could be quite far reaching.

There could be additional sub-objects to the site/object grouping that represent the existing infrastructure and utility and service connections. Like road/objects, these are primarily linear connectors with nodes, and may be similarly represented graphically. Values to include might be conveyance types, speeds, lengths and capacities.

I have not referred to topography as a site characteristic for the simple reason that this is usually a complex three-dimensional entity, and as such cannot be effectively realized in MiniCad. On the other hand, there could be more symbolic representations in 2D, such as points designating crests of hills, or linear elements for swales, or even some sort of designation of preferred views. This would constitute an interesting study of computational methods of diagrammatic site analyses. I have also not addressed the issue of the physical context of the design: adjacent uses, regional issues, etc. These are also information-intensive features of a design problem, and their integration into this design environment could be very useful.

Part 3 - Considerations of Tools for the Realization of a Schematic Model

A designer could obviously make use of such tools as already presented for modeling and information handling. The visualization of relationships of form (even diagrammatical in 2D) and accurate and timely accounting of functional quantities are ongoing *dialogs* that the user engages with the design; the designer suggests, and the model responds. The "greater design realizer" tool that I proposed earlier also engages the designer in a sort of dialog, but one that is much more assumptive and less determining. To take a rough idea of form such as a schematic mass model and to present a "completed result" through automation is in some ways very audacious; from such a low level of idea development, how could simple rules and concepts realize any credible result? I assert that the credibility or completedness of the realization is to be taken at face value; the result is not intended to be "designed" or even to suggest a design. Rather, it is merely meant to be a means to make lumps of characterless masses friendlier, scaled, and more accessible to less imaginative evaluation.

There arises questions of what would make for a suitable representation of this "non-design." What level of detail would be necessary to convey qualities of place and scale, and what level of abstraction should remain to signify the realization's "impermanence?" This too is a subject requiring deeper study than I will attempt to fulfill in this project, and I have to be content working with assumptions. I did attempt to determine some standards of rules and representation for design problems of this sort in M.I.T.'s Design Research

Seminar, Fall 1991. The results of this study were inconclusive, so I will instead work with assumptions I have made.

Degrees of Realization

One of the unexpected lessons I learned about the transformation of schematic modeling objects to a more realized state is that this transformation may occur in stages as the schematic idea evolves. The first demonstration of this partial realization is evident in the command I developed to change the road-line/objects into polygons. By transforming only certain parts or kinds of objects, the designer can continue to model with the still unrealized objects, to find a better "fit" of buildings, spaces (and sidewalks) on a city block. The primary drawback with the transformed road/objects and with the realized transformations in general is that they are not easily modelable themselves. To make changes at this stage would require many more design moves and steps, and it would be just as easy to discard the realized components, return to the schematic ones to edit, and then call a new realization procedure.

Another stage of realization that I found could be useful to a user of this program would be one of simple 3D extrusions of the objects in the design space. By not yet involving the more formal characteristics and relationships of the objects, but instead simply making a 3D massing model of them in the computer, the designer is free to use his or her imagination to understand the spatial effects of the design proposal. This 3D realization has the added advantages of taking far less time for the computer to accomplish, and does not risk being as prescriptive as the proposed greater design realizer. In a sense, however, this stage of realization merely brings the design up to my original conception of the 3D mass-modeling tool, but without editing capability.

Another advantage to the 3D realizer is that it can actually be developed with MiniPascal within the time constraints of this thesis. The procedures are rather simple: For each building/object, the number of floors is read from the object class and multiplied by a factor of floor to floor height (assumed to be 11' for now); the object is then extruded to 3D and resized in the z-dimension to the calculated height. For the ground, the city blocks between the already realized

road/objects are created (using the Clip Surface command mentioned earlier acting upon the realized road/objects), and these block/objects are then extruded and resized to standard curb height, with a subsurface for the entire site the color of road paving to fill the void between the curbs, i.e. the street surface. The space/objects are made into 3D objects but kept at 0' in the z-dimension, and are placed on top of the block/objects. The result is a very suitable 3D massing model for use and viewing at the designer's discretion. (Fig. 3.1)

Greater Design Realization

Much of the information I decided to include in the objects' data fields were considered to serve the additional purpose of informing this particular design tool. For instance, a building object that recognizes itself as ground floor retail and two floors of residential above can now assert this identity in physical form. Likewise with spaces and their defined types and qualities, and roads with their widths and parking values. This information is all essential to making the grossly simplified and abstracted two-dimensional forms of the schematic modeler into elements more recognizable to a casual observer of the design

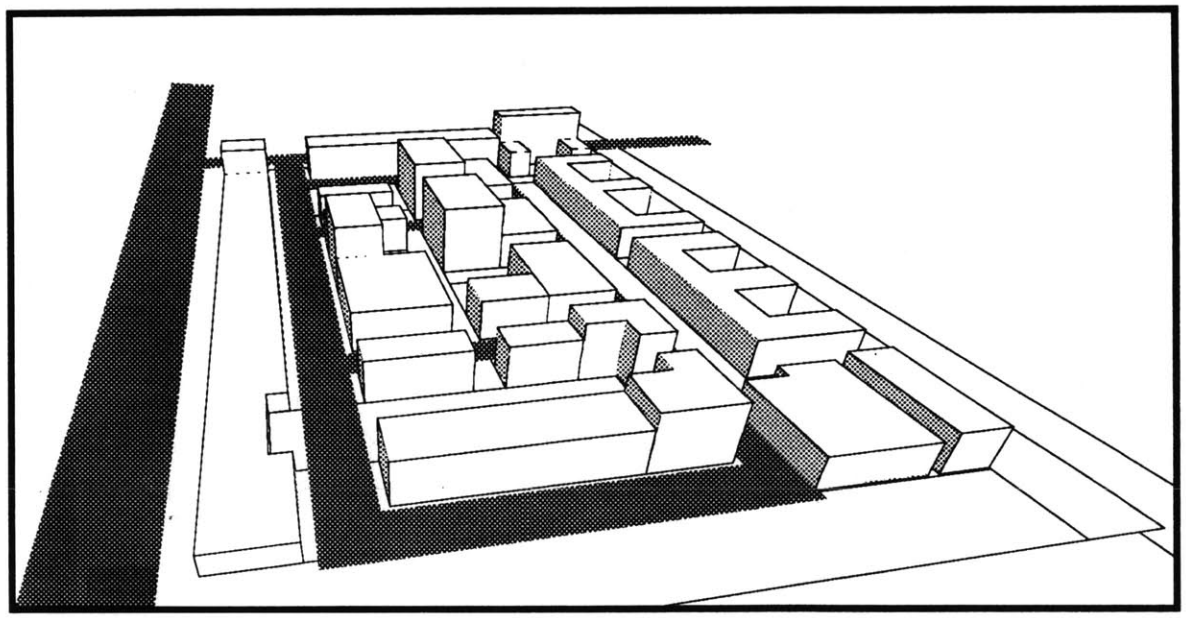


Fig. 3.1 an example of a 3D schematic model in MiniCad

solution. When communicating his or her ideas, it is not enough for the designer to say, "This rectangle is a formal park," and expect the observer to understand the implications of this statement. It is the purpose of the greater design realizer to communicate visually this basic assertion.

(Note: Though the greater design realizer had long been a primary subject of my thesis, I was unfortunately not able to bring these concepts to any fruition. There were some time constraints, but a basic problem had to do with a MiniPascal programming limitation: namely, arrays of data are not able to be passed as parameters to sub-procedures in a program. In lay terms, this means that I cannot write a program for which one set of rules can consider different groups of data. Flexibility is lost. I feel I should add that when not faced with actually creating the procedures I propose, my proposals tend to become quite grand. Nevertheless, I think the ideas I've had for the greater design realizer are valid.)

Dynamic Relationships between Objects

The greater design realization tool that I propose takes the information of each object one step further than simply the assertion of individual object characteristics, and attempts to determine the relationship between adjacent objects. It is at this *interface* that I begin to consider fundamental *rules* concerning elements of large-scale design projects. For example, the retail and residential building just referred to can easily be represented on its own with a conventional facade, but when considering whether it faces a formal plaza or an informal street or whatever, its realization would be affected by this context. The rules concerning the *relationships of objects* is a subject worthy of greater consideration.

For this reason, I have decided to consider the process of greater design realization to be a form of "self-discovery and actualization" of the space/objects in the design; all the new forms and the relationships in the realized model belong to and are a part of the explicitly defined space/objects. This assumption is rather bold, but is based on programming practicality and on a little common sense. I refer the reader to statements I made earlier about what really determines the nature and quality of spaces, concluding that the

surrounding elements of the composition had a major role in this, and that the space in turn influenced the appearance of the surrounding objects. The space *is* the experience of a place, and therefore all emphasis on greater design realization should be placed on this element and its relationships with its surroundings.

Data Handling Strategy

Considering spaces and greater design realizing in this way would make the command procedure easier to develop, since the really intensive information gathering and rule application processes would have to deal with only one particular type of object, calling therefore for only one set of rules (albeit, a complex set). The first step of the process would be for each space/object to gather the information concerning the objects surrounding it; a sub-procedure could read the perimeter of the space/object at intervals (say, at 5' or 10' depending upon the snap-grid setting the objects were created in), looking for building/object information regarding begin and end points of its facade, its ground level and upper story functions, and height. For adjacent roads, the space/object would inquire about vehicle capacity (lanes), the noise level of which would have a determining effect on that edge. (This assumption does not deal with the question of roads actually being *part of* a space; this question proved too difficult to resolve with the space/ and road/objects as defined here, and would require further study.) The position of adjacent space/objects would also be read, which would influence pedestrian movement patterns through the space at hand. The space/object would also compile information about its "group" of focal points and/or arcades, if any.

Having compiled all this data about adjacencies to a single space/object into arrays, the program would then take this information and proceed to apply rules for determining some basic spatial dynamic properties for the space/object and its surroundings. (At this point is where MiniPascal failed to live up to expectations. I could not apply any ideas I had about these rules to an actual example without great time and effort, and therefore the following rules are still only theoretical. The illustrations shown are not examples of any real procedure, but merely graphic representations of the ideas discussed.)

Type Definitions and Qualities

By defining the various objects in the design as specific types or type combinations, I am using the idea of involving prototypes for design. Prototypes are one of the issues concerning computational design processes, as mentioned in Part 1, and their usefulness has become apparent to me. The prototypes I am considering for the greater design realization of a large-scale development project are in some ways similar in nature to Christopher Alexander's Pattern Language in that they deal with discrete *types* with variations according to scale and proportion. I am also reminded of shape grammars to the extent that the realization of these specially-adapted types involve construction from elements according to rules.

The prototypes that I propose for greater design realization are intentionally standard and unimaginative. This repertoire could in time be expanded to include any number of examples. For the data fields set up for space/objects in the modeler, I allowed three fundamental types: plazas, parks, and streets; and two fields for "quality": formal and informal. This then calls for six prototypes to consider. I must also be explicit about the sort of place I am assuming this design to be when completed; namely, a "traditionally urban" place, densely built and buildings of up to five stories in height. I imagine the repertoire of prototypes and prototype rules could eventually accommodate standards for suburbs, small towns, big cities, etc.

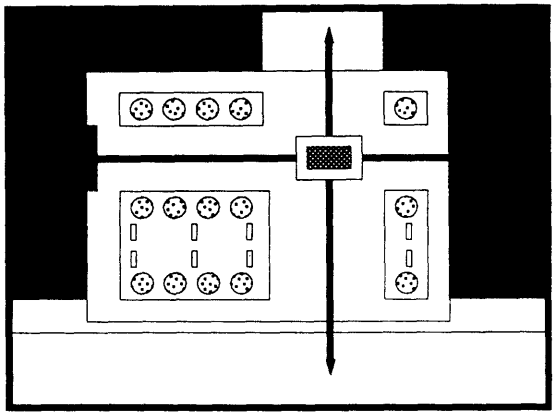


Fig. 3.2 prototype of a formal plaza

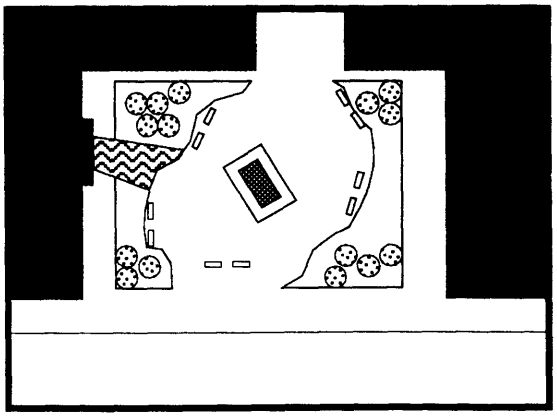


Fig. 3.3 prototype of an informal plaza

I have defined "plaza" for these purposes to be a public open space characterized by paved surfaces, small trees (if any) in planters (submerged or not), and seating areas. Building facades are prominently featured in plazas, and adjacent roads are often buffered by trees or other plantings. A formally arranged plaza usually incorporates a rational geometry related to surrounding buildings, setting up axes and focal points. Trees are in rows or neat groves and set away from the axes and foci. (Fig. 3.2) An informal plaza is more rare, but usually tries to recognize natural use patterns and does not try to have an overriding formal logic or geometry. No elements need be aligned to the surrounding objects, and pedestrian movement patterns are dominant. (Fig. 3.3)

A "park" is usually of larger scale than a plaza and is characterized by unpaved grassy areas, medium-to-large trees, and pedestrian paths edged by seating. Often there will be a node or focal point within a park. I tend to consider a more ideal park to have more trees around the periphery and open grassy areas towards the middle. I found it helpful to think of parks that are directly adjacent to a building as having a path along that edge. Larger parks would have a more complex arrangement of nodes, trees and grassy areas, but I will only consider smaller parks for now. A formal park has a focus with eight radiating paths (in a rectilinear design scheme), four perpendicular to the park's edges and four diagonal to its corners. (Fig. 3.4) An informal park resembles those in the English landscape tradition, with meandering paths connecting corner and side entries and interior nodes or focal points. (Fig. 3.5)

I have intentionally avoiding using the term "street" when referring to roads in the schematic design. The connotations of "street" involve a sense of place,

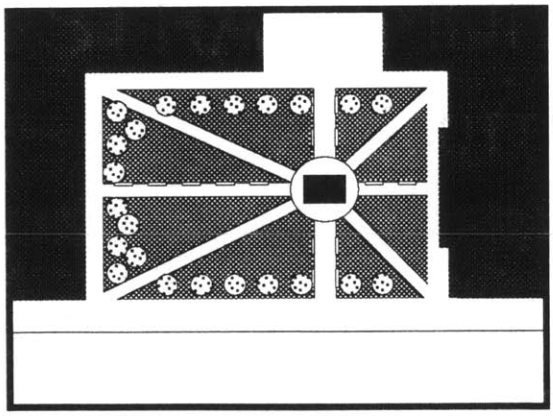


Fig. 3.4 prototype of a formal park

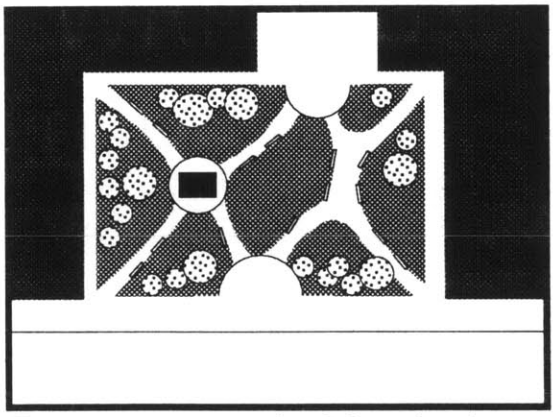


Fig. 3.5 prototype of an informal park

and for the purposes here are pedestrian-oriented. A street is directional, a conveyance, but with character and *life*. Surfaces are paved, and there is usually a row or two of trees reinforcing the directionality. (Fig. 3.6) For reasons of variation, trees on east-west running streets would only be to the northern side of the street, where more sunlight is available and shade desirable in the summer. On north-south running streets, trees could be on either or both sides and more widely spaced on center. (Fig. 3.7)

A variation of the street as defined here is the sidewalk, which I consider different from a street only in that a street is bounded by two buildings or rows of buildings, while a sidewalk is bounded on one side by a road. Though not implemented, sidewalk/objects could be created and recognized as objects during greater design realization by using the Clip Surface tool to pick up any areas not yet covered by an object. If the object is long and narrow and bounded on one side by a road and the other by buildings, then it would be defined and realized as a sidewalk.

Parking areas could use realization too. There are already programs for parking lot layouts and landscaping which would effectively address this issue as a component to this program.

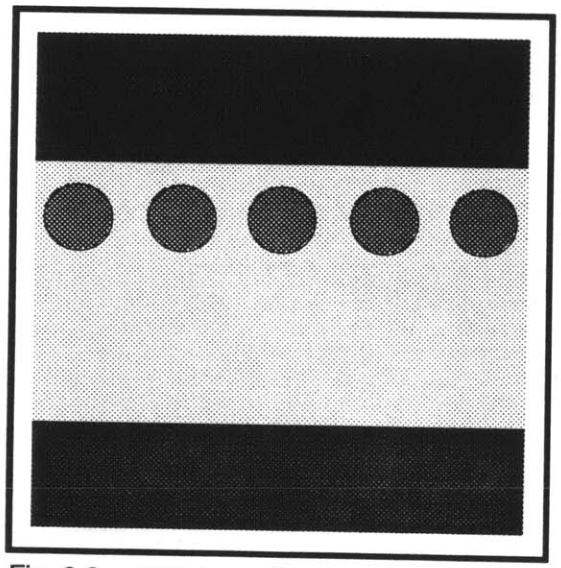


Fig. 3.6 prototype of an east-west street

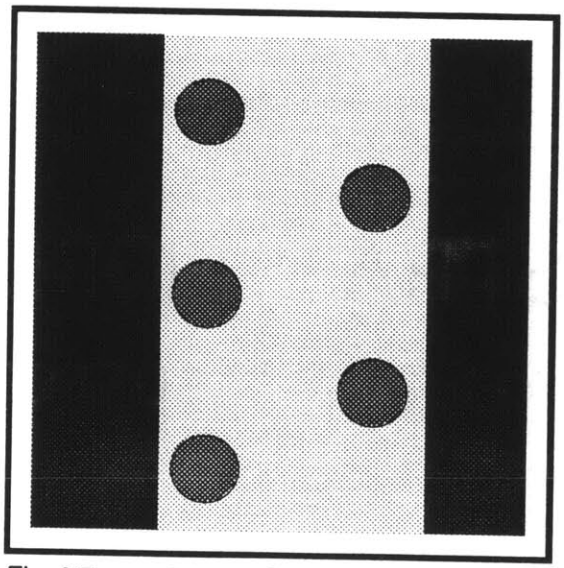


Fig. 3.7 prototype of a north-south street

There must also be prototypes for the building facades. Considered simply, the ground floor of facades facing spaces often are storefronts, having a high percentage of glazing whether retail or office. Awnings are common in this urban setting. The upper floors are repetitious, and there is some form of top "where the building meets the sky," such as a cornice or roof overhang. Facades can have a rhythm horizontally, which may be broken or articulated for an entry/focus or to enhance the dynamics of the space it fronts. Some buildings may have a strong focal point in the form of a tower, which is intended to terminate a vista. Many buildings employ symmetry to the entire facade; I feel that this grand symmetry should be reserved for important or dominant buildings in a space. Smaller symmetries such as for entry/foci are still encouraged.

Determining a Dynamic

I admit an intentionally *designed* space would probably much better represent spatial dynamics than these very generalized programming tools. Nevertheless, I will count on two fundamental principles of spatial definition to propose a procedure for assessing some basic qualities of a space: axes and foci. By determining the axes and foci of a space, the dynamic can be enhanced. I feel that by applying rules based upon derived spatial dynamics, more sophisticated instantiation of prototypes is possible, which I feel is preferable to simply instantiating cookie-cutter plans.

The interplay of the building/, space/ and road/objects surrounding a space are the determinants of axes and foci in that space. A building can front a space in several ways. (Fig. 3.8) Which of these in fact is the case, coupled with which type the space is defined to be, has a bearing on what dynamic is derived. The rule I would use is that if an individual building facade occupies more than fifty percent of an edge of a space and does not extend beyond the space, then an axis drawn perpendicular from the midpoint of that facade into the space is a candidate for major axis. (Fig. 3.9) If that particular axis is chosen to be primary, the origin building would have a symmetrical facade. With adjacent spaces, the coinciding edge segment is a transition of movement from one space to the next, and as such the perpendicular of the midpoint of that segment is another

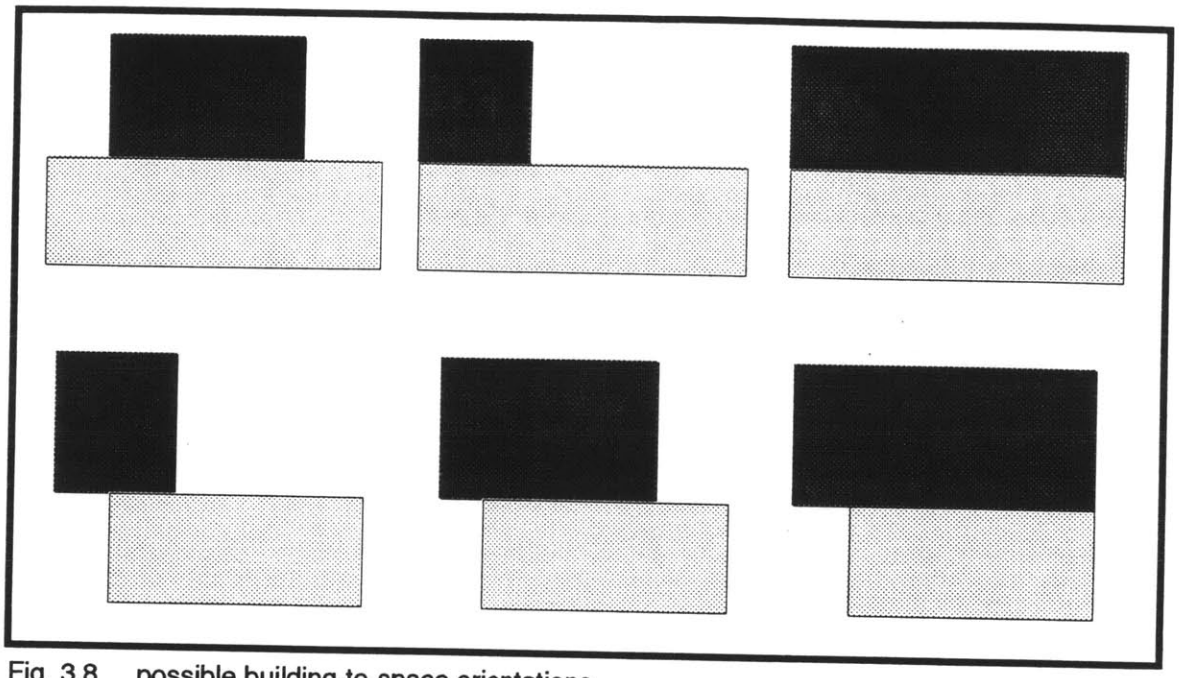


Fig. 3.8 possible building-to-space orientations

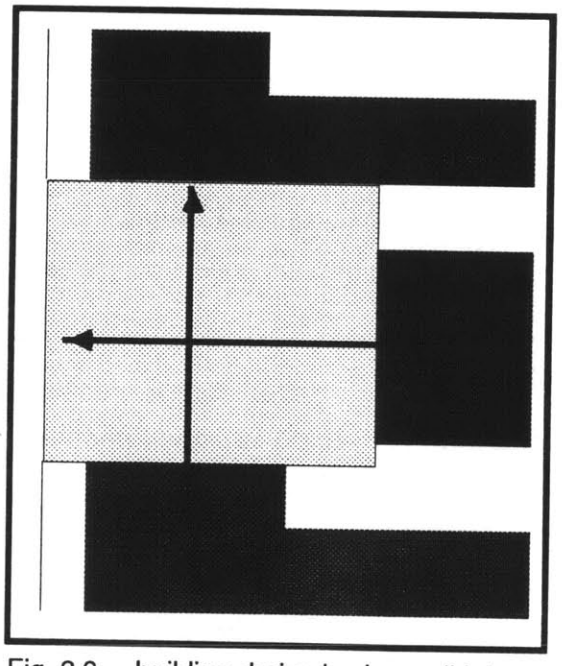


Fig. 3.9 building-derived axis candidates

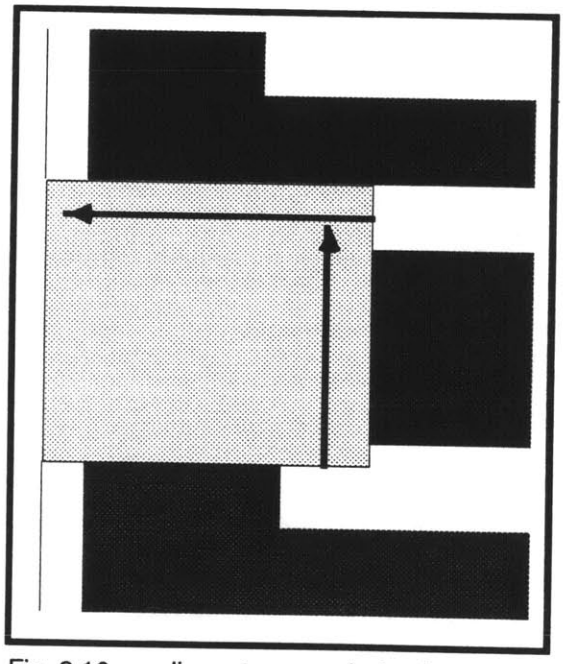


Fig. 3.10 adjacent space-derived axis candidates

candidate for major axis. (Fig. 3.10) Where the axes intersect, these are the candidates for focal point. (Fig. 3.11) For reasons of simplicity, I would suggest that the focal point candidate nearest to the center of the space be selected for realization. If an axes from an adjacent space does not intersect this chosen focus, then I would suggest some articulation of the facade opposite. (Fig. 3.12) These rules are somewhat arbitrary and may produce strange results; I cannot tell for sure without actually writing the procedure. I am sure that if applied, some adjustment and additional rules would be in order.

The tool already mentioned for creating space/objects allowed for user designation of focal points. This element when defined would do much to determine the dynamics of a space and would render most further computational determinations of dynamics unnecessary. But the greater effects of the dynamics on a space, as in facade treatments and the placement of trees and seating, still would need to be resolved.

Constructing Realized Spaces

If the task of realizing spaces involved simply inserting a prototype and resizing it to fit the special case, the procedure would be very easy to write. But by

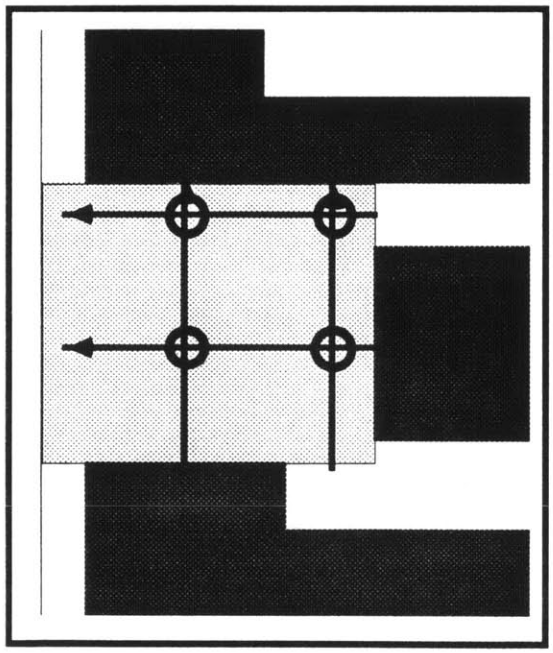


Fig. 3.11 focal-point candidates

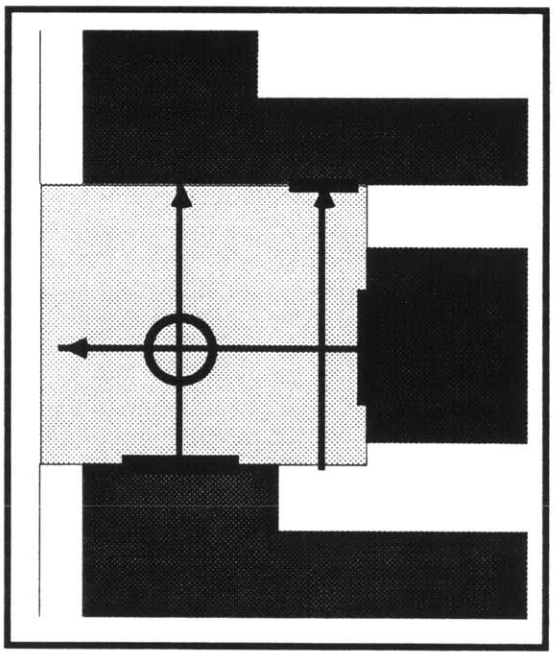


Fig. 3.12 primary spatial focus and other edge foci

involving possible variations in scale and spatial dynamics to influence the result, each realized space is necessarily reconstructed by a set or rules. The prototypes, therefore, *are* the rules for this construction process. The first element laid down in this process would be the ground plane, which would be done differently depending upon what type of space is being constructed: plaza and street ground planes would cover the entire space uniformly and represent paving; park ground planes would represent grass and not extend to the full edge of the space (leaving a perimeter walk). Next, the primary focus would be instantiated (in the case of plazas and parks) where determined by the dynamics procedure or by the designer. Paths would then be created, connecting focal points and adjacent spaces, and corners in the case of parks. Trees and benches would then be inserted, by rules according to the prototypes illustrated already. (Fig. 3.13)

The construction of facades for the surrounding buildings would require a different set of rules, while still adhering to the spatial dynamics already determined for the space. The study of shape grammars and building facades

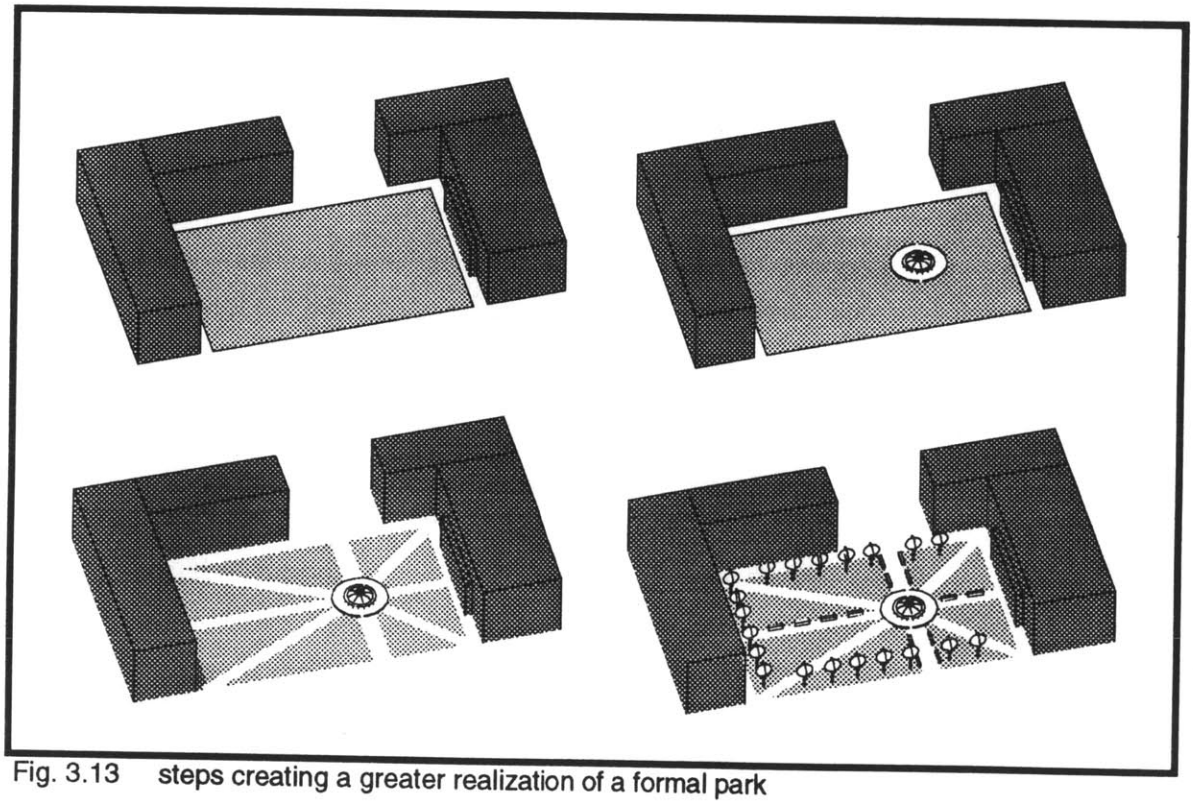


Fig. 3.13 steps creating a greater realization of a formal park

would have particular relevance to this procedure. For simplicity's sake, I have worked on a method of inserting and positioning specific building floor and bay types. The distance from the begin and end points of a building edge is divided into structural bays of 20' to 30', with off-rhythm bays allowed at edge focal points. Each bay facade is then constructed, inserting a ground floor element indicative of the use of the ground floor, and a series of upper floor elements indicative (by fenestration) of the uses of the upper floors. (Fig. 3.14) The building top or crown is then placed. I have developed facade element in three generic "styles", and many more would be desirable. More formal or important buildings could have additional rules regarding bay spacing and the use of crowning elements.

It seem like there would be some need for randomizing in this greater design realizer. What style type, for instance, or what bay rhythm combinations to use are rather arbitrary questions when realizing building facades. Tree sizes, placement and spacing in formal and informal spaces would be others, as well as path widths, bench placement, and focal point type (statue, obelisk, or fountain).

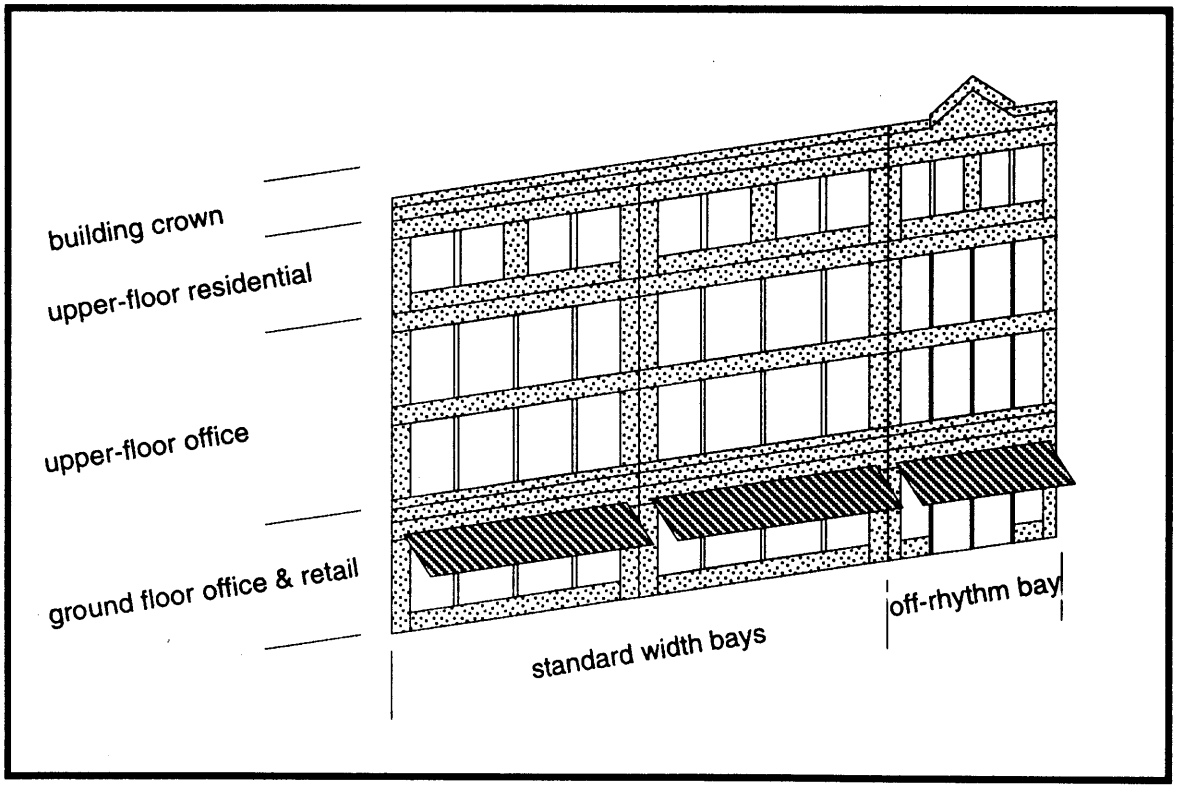


Fig. 3.14 building facade symbol elements in a typical arrangement

Instantiating with MiniCad Symbols

This strategy of inserting and repositioning elements for realization is handled in MiniCad by the use of "symbols." MiniCad symbols can be created quite flexibly and held in libraries of folders and sub-folders, which can be easily exchanged among files. I would imagine this system of libraries and folders would be very helpful for organizing prototype information and improving automation techniques for using these symbols. These symbol elements can be three-dimensional, and they require less computer memory than regular file elements. Two drawbacks that I have found are the inability to resize or rescale symbols and some problems regarding the orientation of symbols in 3D. These

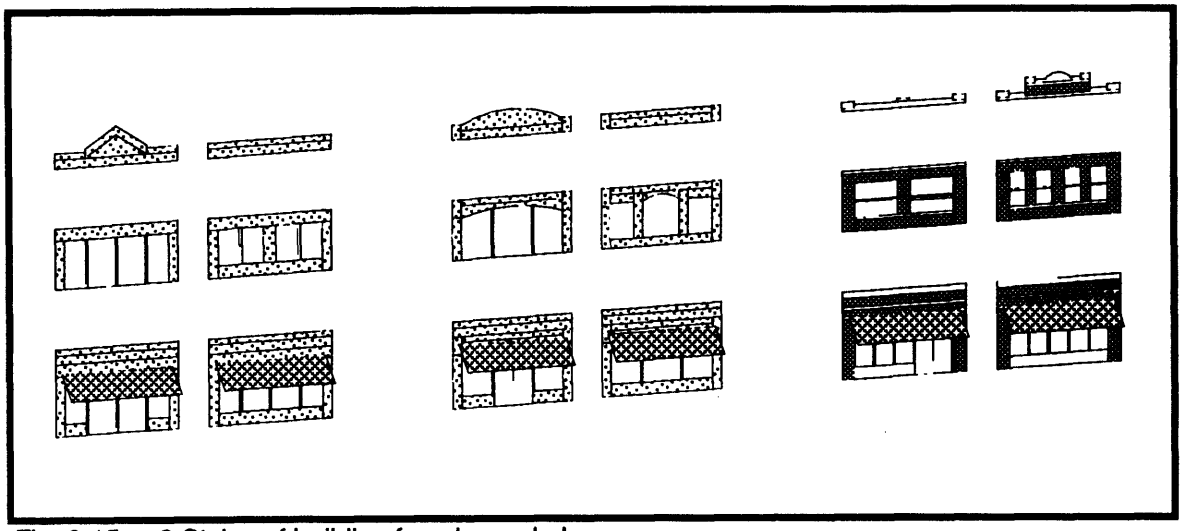


Fig. 3.15 3 Styles of building facade symbols

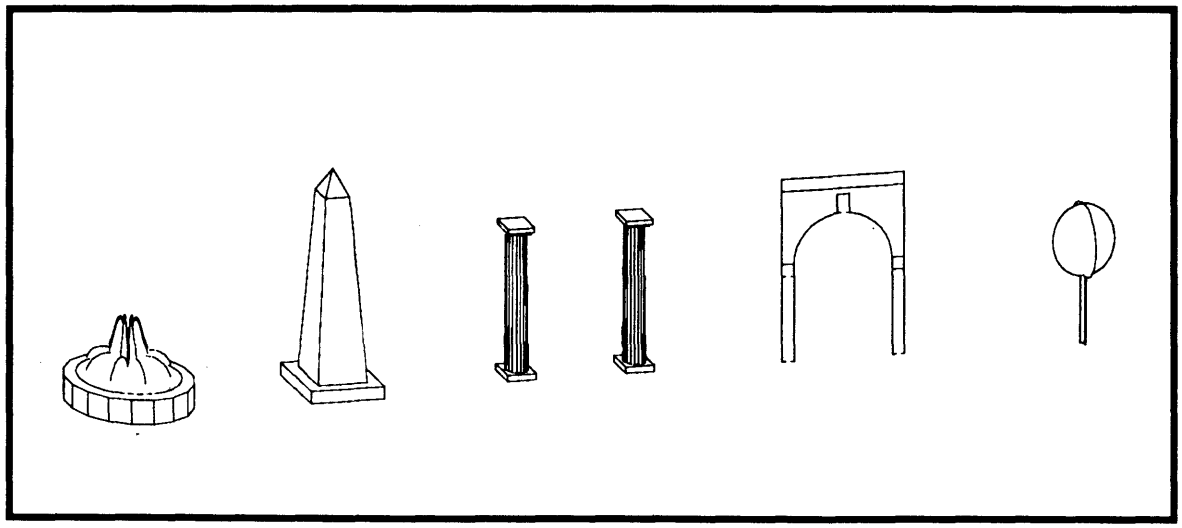


Fig. 3.16 some examples of spatial element symbols

problems can be overcome, but the former would require "de-symboling" the objects and greatly increasing computer memory needs. Figures 3.15 and 3.16 illustrate examples of the symbols I created in MiniCad for use in the greater design realizer.

Viewing Possibilities

The end result of the greater design realizer is a new model of the design, distinct from and in addition to the schematic model. This 3D model can be saved in another file for subsequent use and then deleted from the original design file. The original file would revert to its 2D schematic level for further consideration and design moves, and for later design realizations. Over the course of the design evolution, several models could be saved in this manner, constituting a record of the design's progress.

The new 3D model is intended primarily for viewing and visual testing of the current design solution at intervals in the design's evolution. As it is, the user would need to be familiar with MiniCad 3D viewing capabilities, which aren't too sophisticated or flexible in my opinion. I was not able to get to the "visualization tool" that I proposed at the beginning; it would have been particularly useful given MiniCad's limitations. For this, I would have to blame time constraints as well as the lack of such a model to view anyway.

Part 4 - Scenarios for the Application of these Tools

The tools I have presented and the work I have done in developing and implementing them have been a valuable exercise in understanding issues concerning the design of large-scale urban environments and computer methods and processes. But a prevailing issue that goes beyond the bounds of simple exploration is of what use are these tools to real design projects. Who might use them, how, and to what end? In this section I will offer suggestions of ways these these tools be used and illustrate a some possible scenarios using them for the design of a fictitious project.

Two Design Examples Using these Tools

To demonstrate a few ways in which these tools may be used to design a large-scale project, I will use an imaginary example of a suitable project, greatly simplified to accommodate the limitations of the tools as thus far developed. I am illustrating two different ways of approaching the same design project with the same tools, demonstrating that some degree of flexibility is inherent in this design environment. The "site" (Fig. 4.1) is at the junction of two existing roads: a six lane highway to the north and a four lane road to the east. The south is bordered by a small river. The 425' x 775' rectangular site has a setback of 15' on the north and 30' on the east and west.

First Example Design

The first example design begins with a proposed structure for the public spaces. (Fig. 4.2) The designer here proposes a promenade along the river with two parallel pedestrian streets inland, using the command "Define Spaces" for creating these space/objects. Three pedestrian avenues lead from the river to differently configured plaza space/objects along the second parallel street. Next, the designer inserts road/objects with the "Roads" command in a straightforward manner to allow access through the site, (Fig. 4.3) running parallel to pedestrian streets in some places, and independently in others. The primary road is a loop, with two single lane streets cutting north-south across the loop. Happy with this configuration the designer decides to consider the building program: 185,000 total gross square feet, with 15% retail, 45% office, and 40% residential. These figures are entered with "Prgrm Pcts." and are graphically represented in the reserve space above the design. (Fig. 4.4) The designer also chooses to view the roads at their actual width, realizing them with the "Real Roads" command.

At this point the designer decides to make adjustments, adjusting the space/objects around the roads and fitting building/objects between these already inserted objects. The designer works with single-level building/objects to design the ground level activity, focusing all of the retail space around the plazas along the second street from the river. (Fig. 4.5) The remaining ground level within the loop road is filled with office uses, as is the area between the loop road and the existing road to the east. The designer is careful to leave gaps between building/objects and realized road/objects for at least minimal sidewalks. Using "Parking Areas", two sections are designated for parking (diagonal hatch, Fig. 4.6), a long double-level structure between the north section of the loop road and the main highway, and a three-level structure in the south-west corner. The river edge away from the promenade is devoted to residential use. Using most of the remaining objects in the reserve space, the designer then stacks upon the ground-level configurations (Fig. 4.7), mostly office up to 3 to 5 stories within the loop, and all 3 story residential along the promenade, with a 5 story "tower" on the eastern end. Having utilized all the elements into a schematic fit, the designer can then invoke the "3D Extrude" command to create a 3D mass model of the design as it is so far, to be viewed

in axonometric or in perspective. (Figs. 4.8 and 4.9) (Note: There was no example made for "greater realization".)

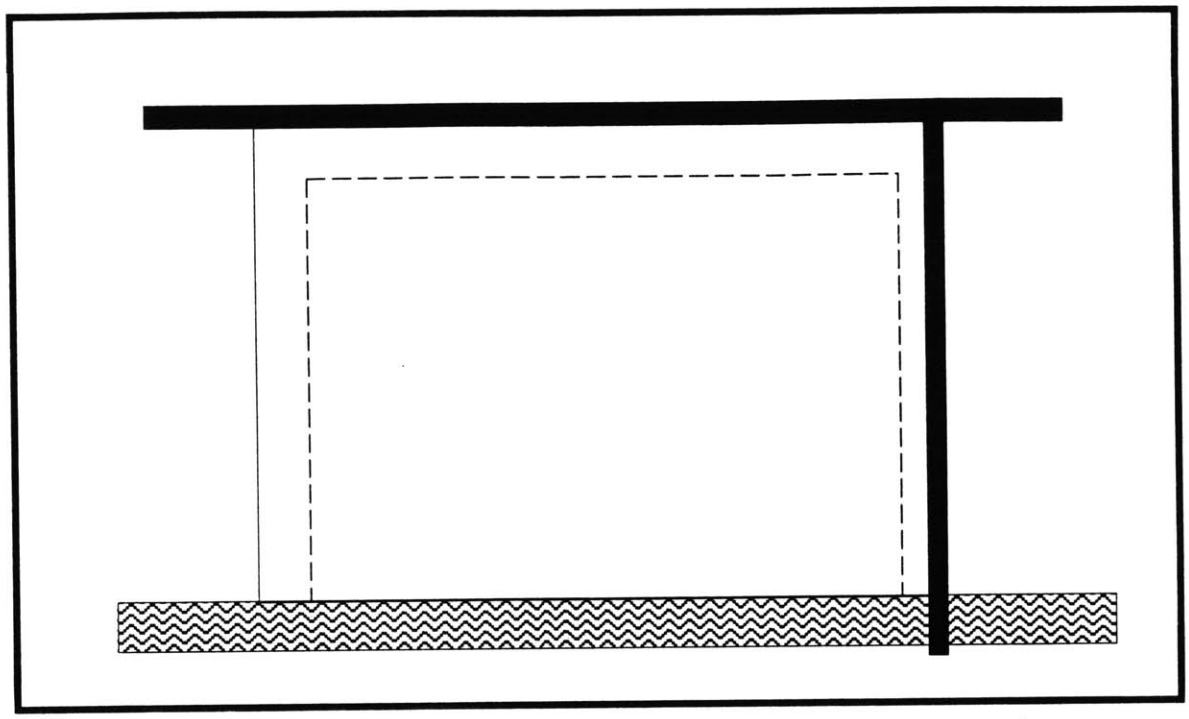


Fig. 4.1 fictitious site: 425' x 775', with roads to north and east, and river to south

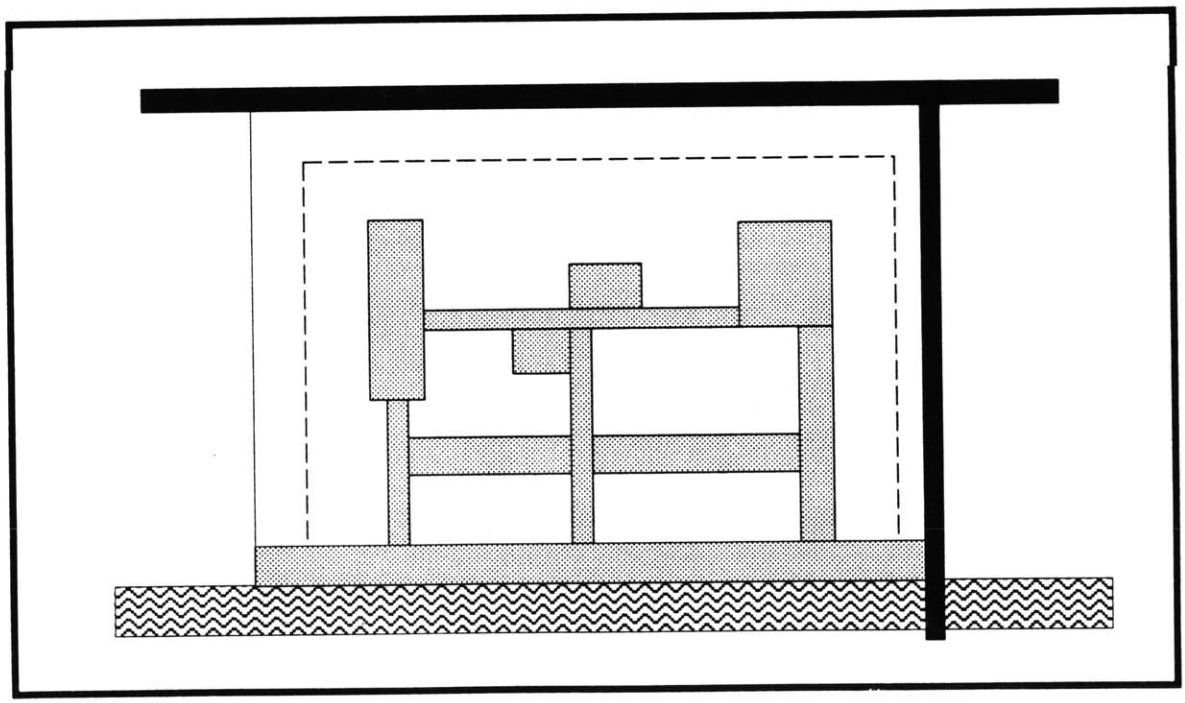


Fig. 4.2 example 1: proposed spatial structure for development

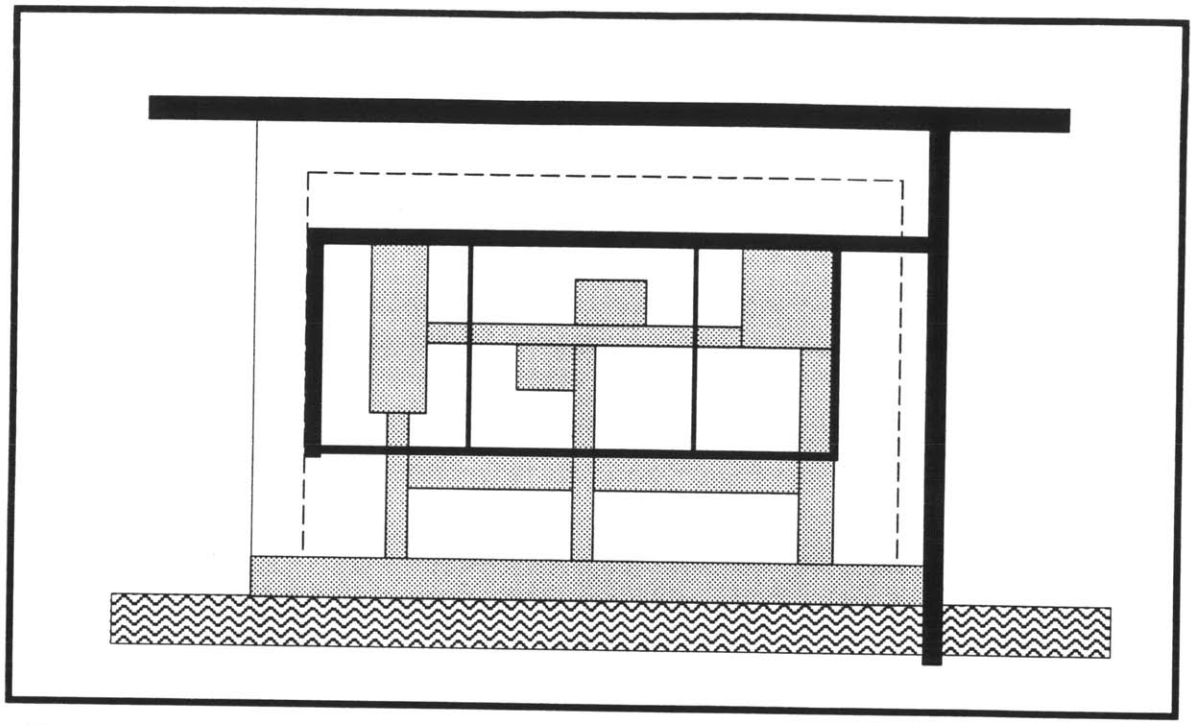


Fig. 4.3 example 1: incorporation of road/line objects

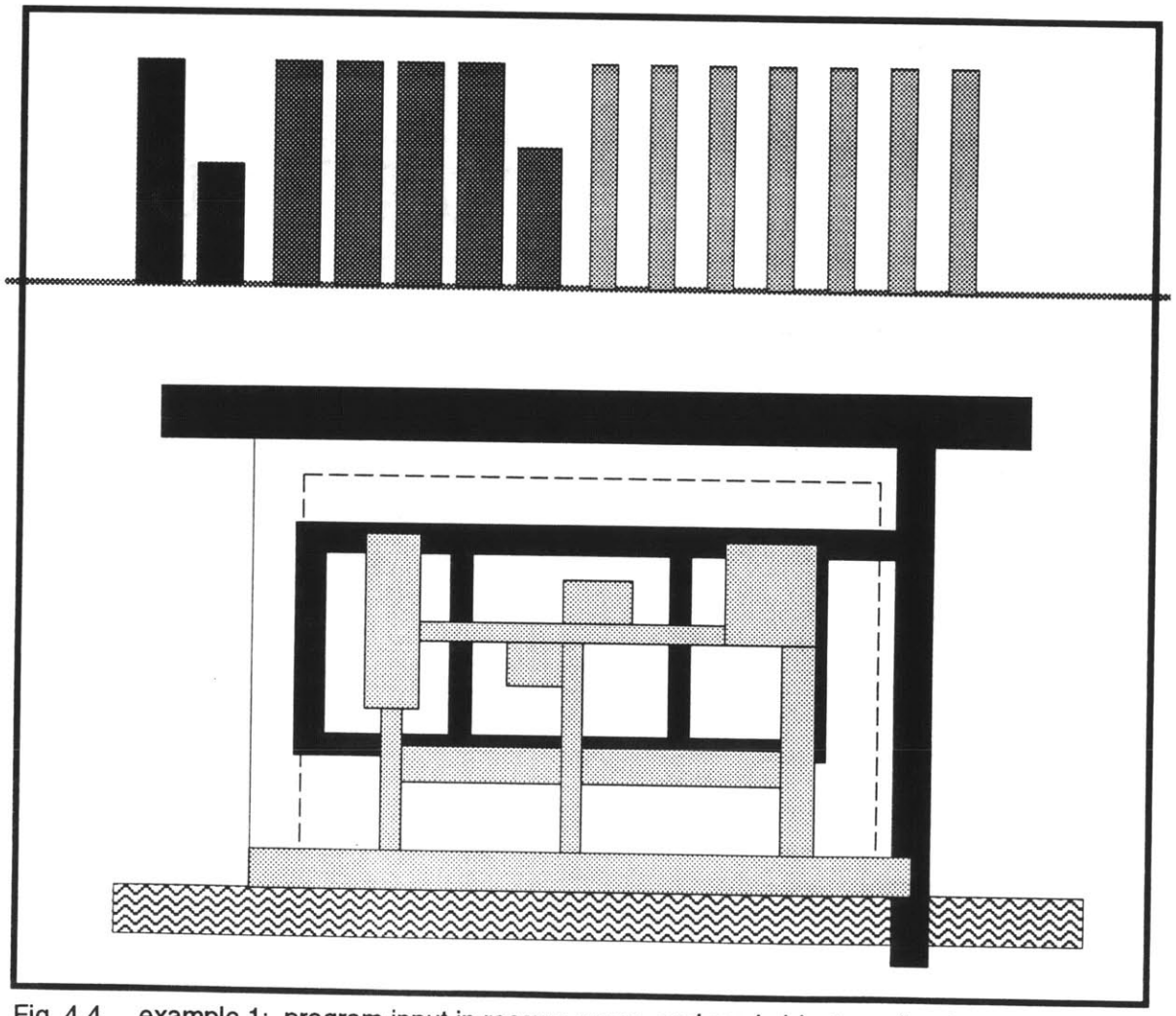


Fig. 4.4 example 1: program input in reserve space, and road objects realized

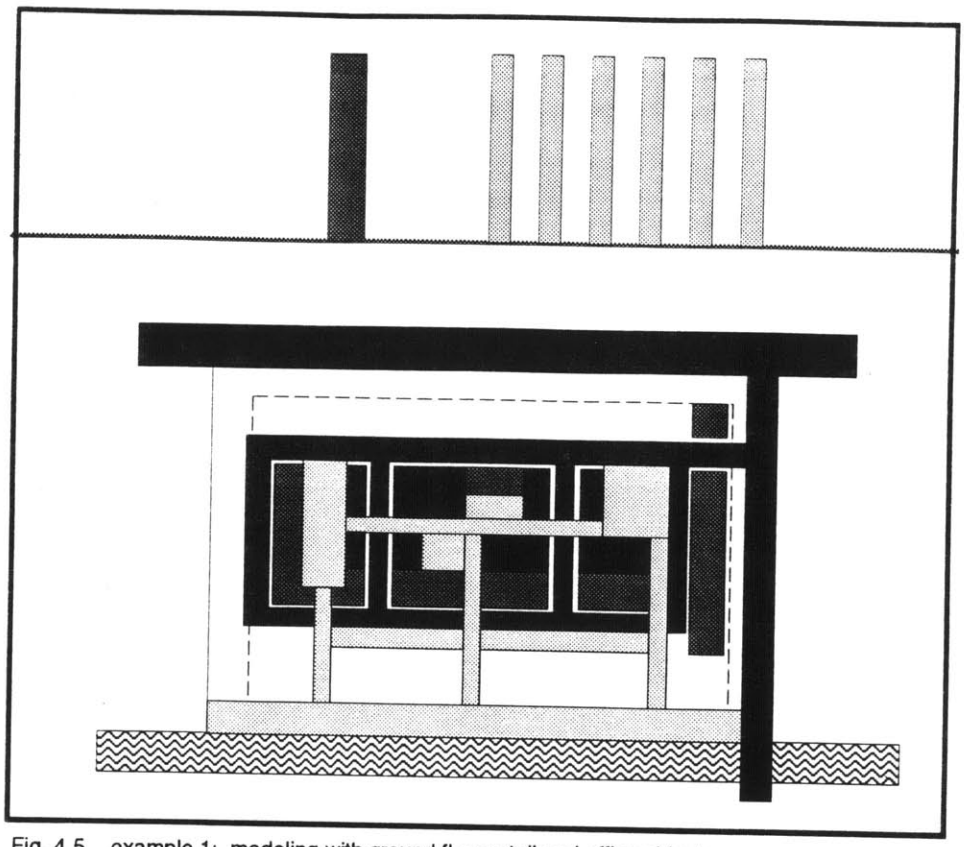


Fig. 4.5 example 1: modeling with ground floor retail and office objects

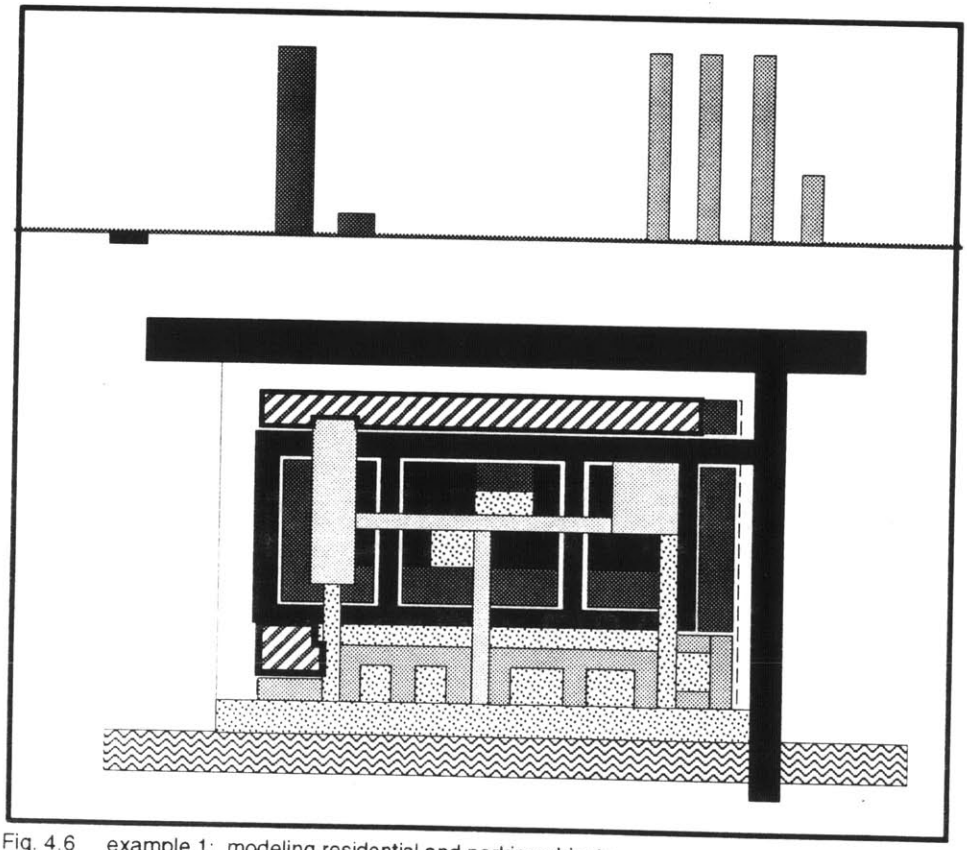


Fig. 4.6 example 1: modeling residential and parking objects

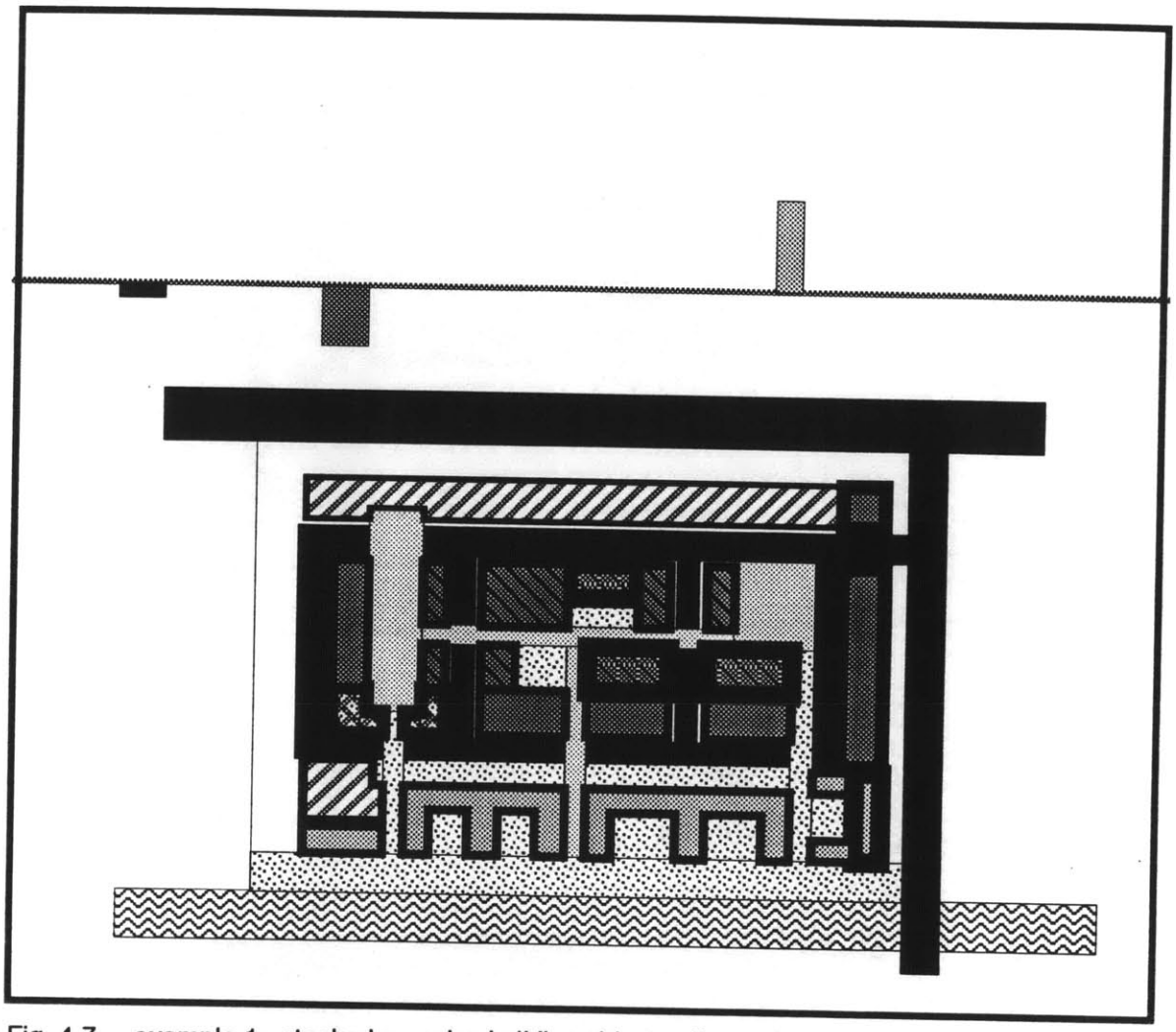


Fig. 4.7 example 1: stacked complex building objects - line weights reflect building heights

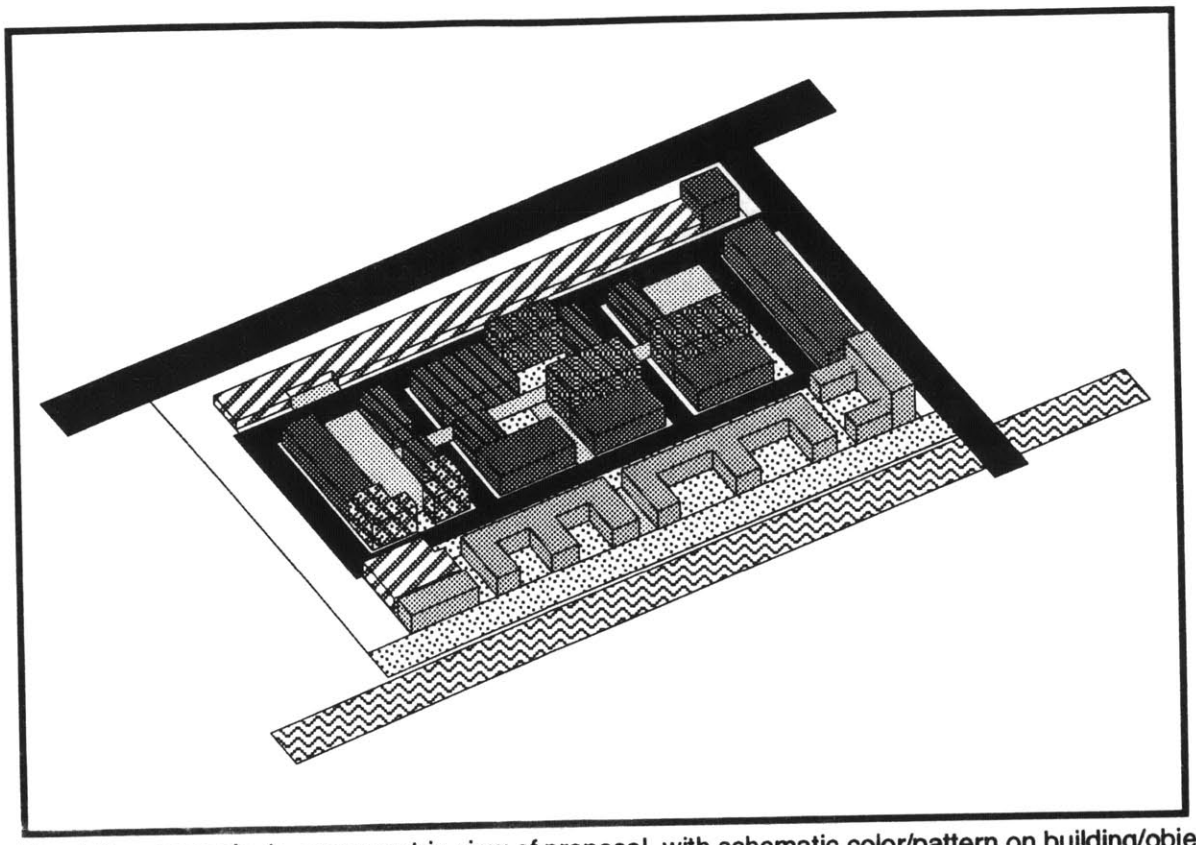


Fig. 4.8 example 1: axonometric view of proposal, with schematic color/pattern on building/objects

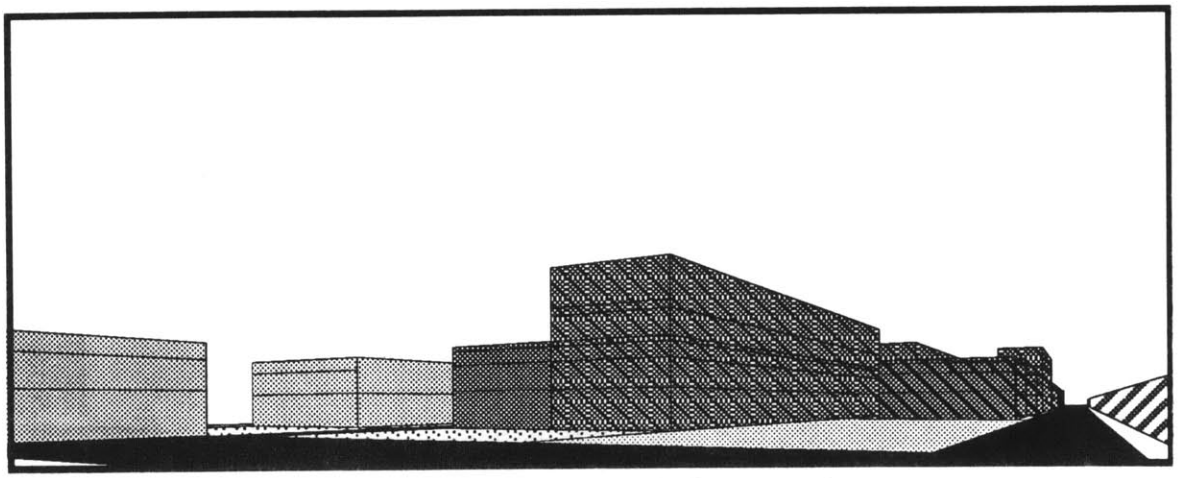


Fig. 4.9 example 1: perspective of schematic model - view at north-east entrance plaza

Second Example Design

The second example uses the same tools but in a different order, creating different sorts of design stages. (I think that this second approach may be how a planning professional might use these tools.) Using the same site, the user first decides to input the building program, the same program as in the previous example. Also initially, a road network is created through the site to serve as a functional framework for the design. (Fig. 4.10) The user has determined that s/he wants to use three basic building types in the project: two-story residential, two-story office, and three-story office with ground floor retail. The user stacks the building objects in the reserve area to created these distinct types from the input program. (Fig. 4.11) The user has also decided to shift the road network to the southeast from the originally created position (This is done while the road/objects are still lines and not "realized" polygons.). The user continues to work with the building objects while still in the reserve space, creating a single large mass of the retail/office objects, and breaking down the residential objects into uniform, smaller units. (Fig. 4.12) The road network is again shifted, this time to the center of the site.

The user next places the building/objects around the site, with the three-story mass in the middle flanked by office objects and the residential blocks along the north and south edges. (Fig. 4.13) Parking areas are created on the east and west. Within this arrangement the user then works to create a fit of all objects in the design space. (Fig. 4.14) The road objects are realized to their true dimensions, and the buildings within the loop are split and arranged within these blocks, with a pedestrian street defined down the middle perpendicular to the river. A large park area and promenades are defined along the river, and the residential units along here have to be clipped to fit. The resulting residual residential area is combined and stacked to three stories and inserted along the north edge to serve as visual terminations to the smaller north-south roads.

Figures 4.15 and 4.16 illustrate one more technique for using these modeling tools, whereby one could create a single large mass according to some form of zoning guidelines, and then "carve" a network of public spaces out of this mass. This is a way of maximizing allowable built area, not designing subject to any predetermined building program. In this way, it would be attractive to financial interests while allowing a sculptural approach suitable to the designer.

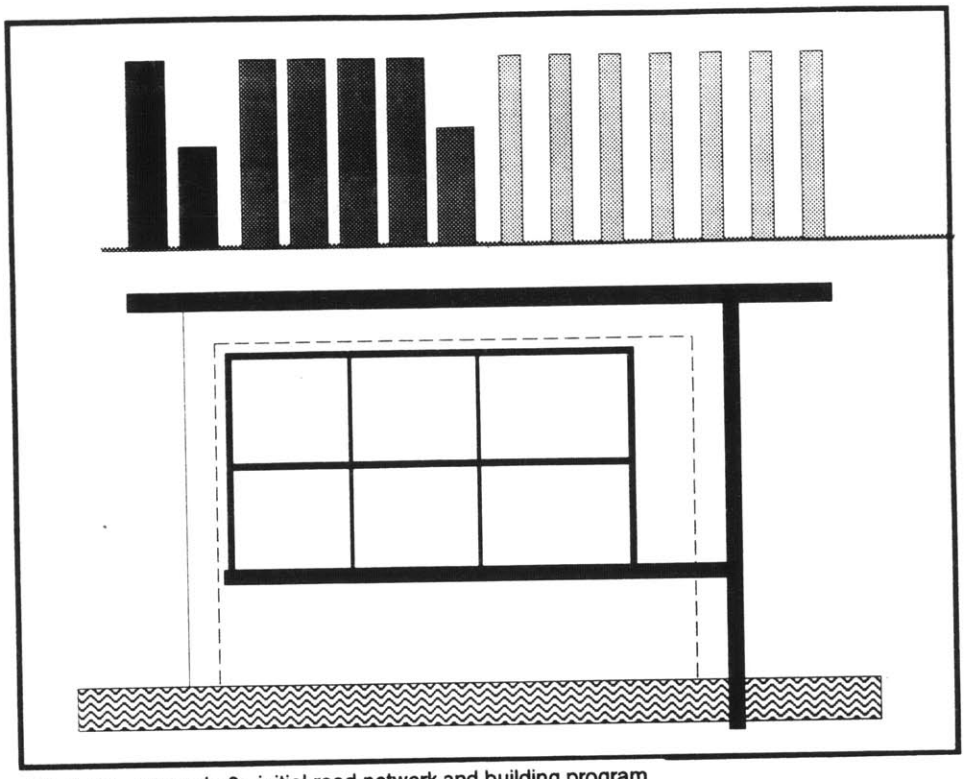


Fig. 4.10 example 2: initial road network and building program

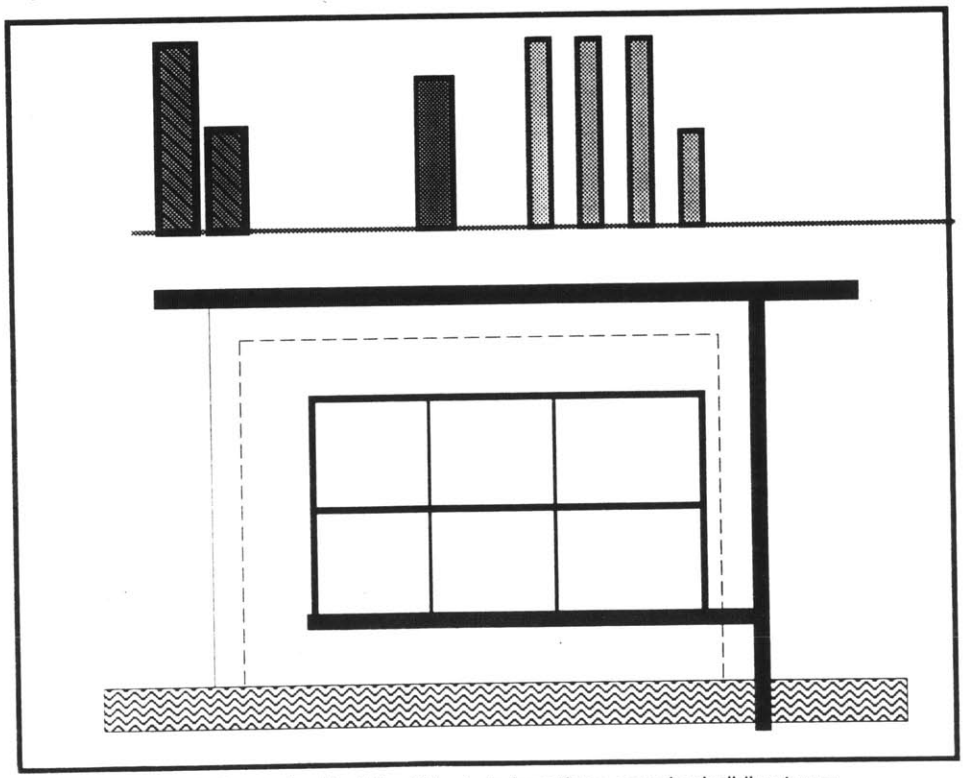


Fig. 4.11 example 2: stacked building/objects to form three complex building types

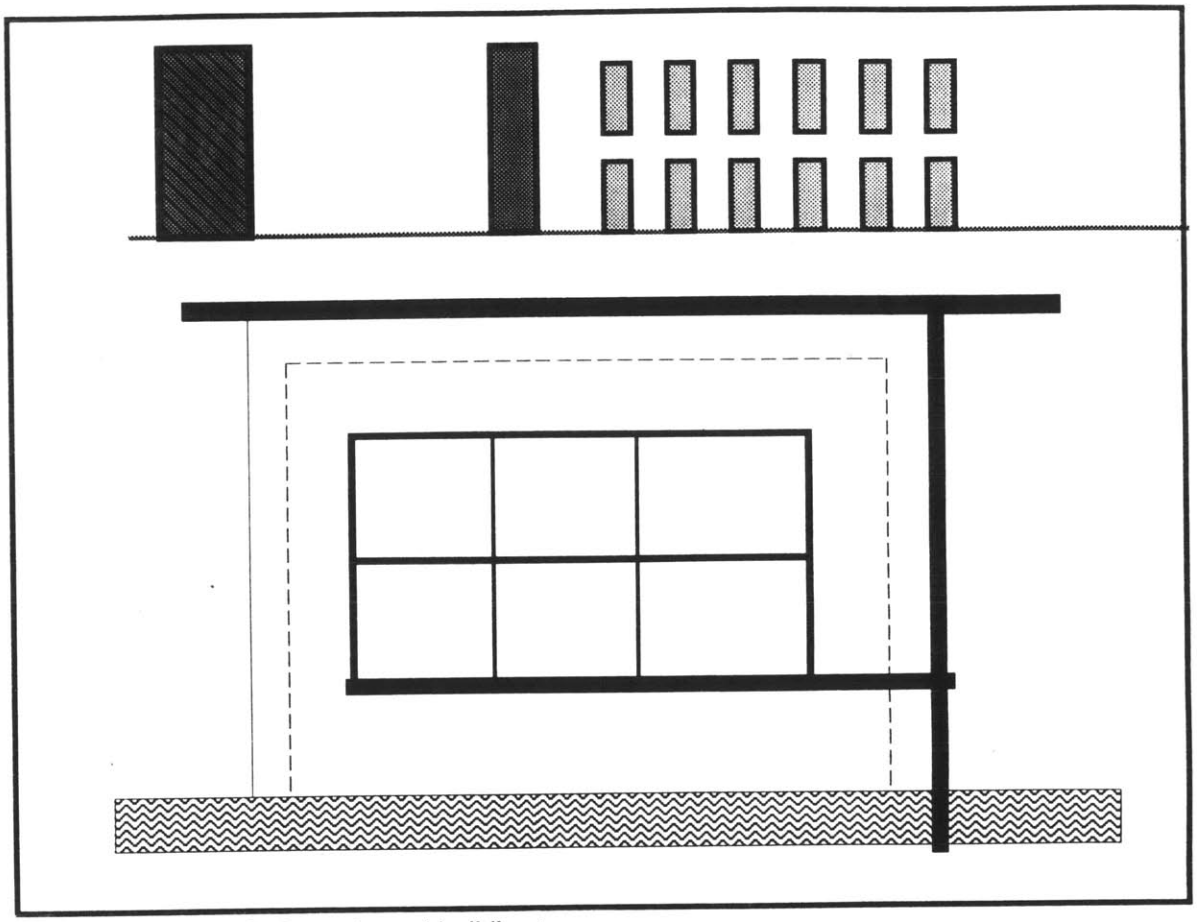


Fig. 4.12 example 2: reshaped building types

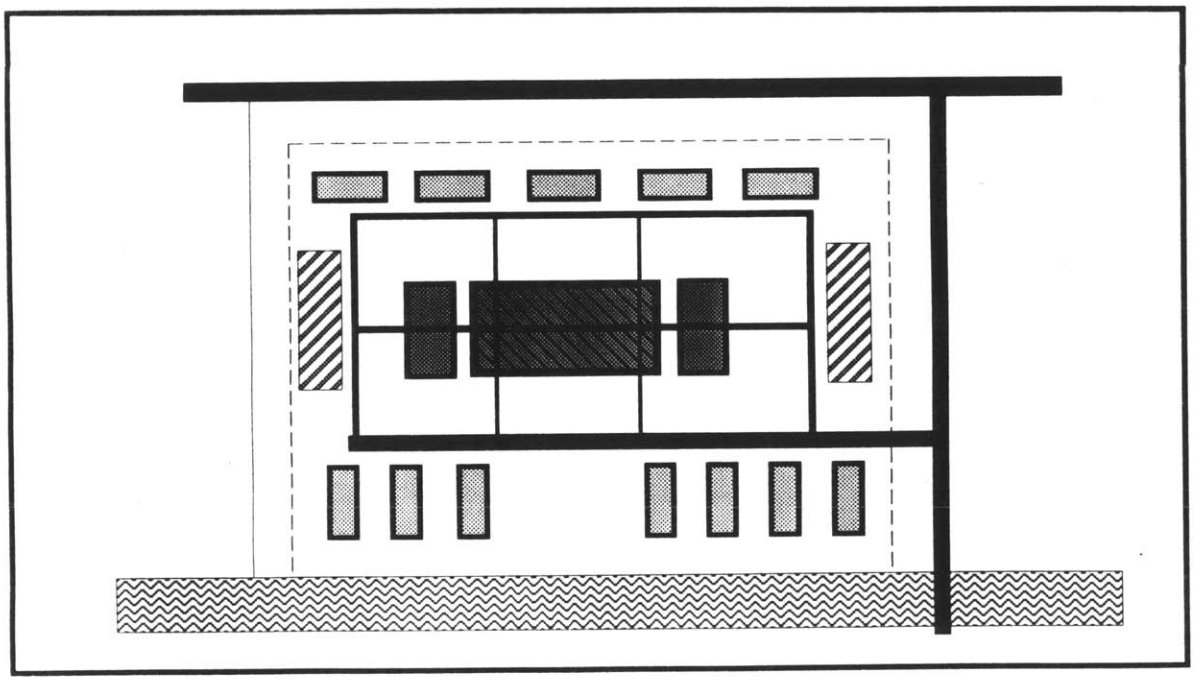


Fig. 4.13 example 2: building type layout on site with parking areas

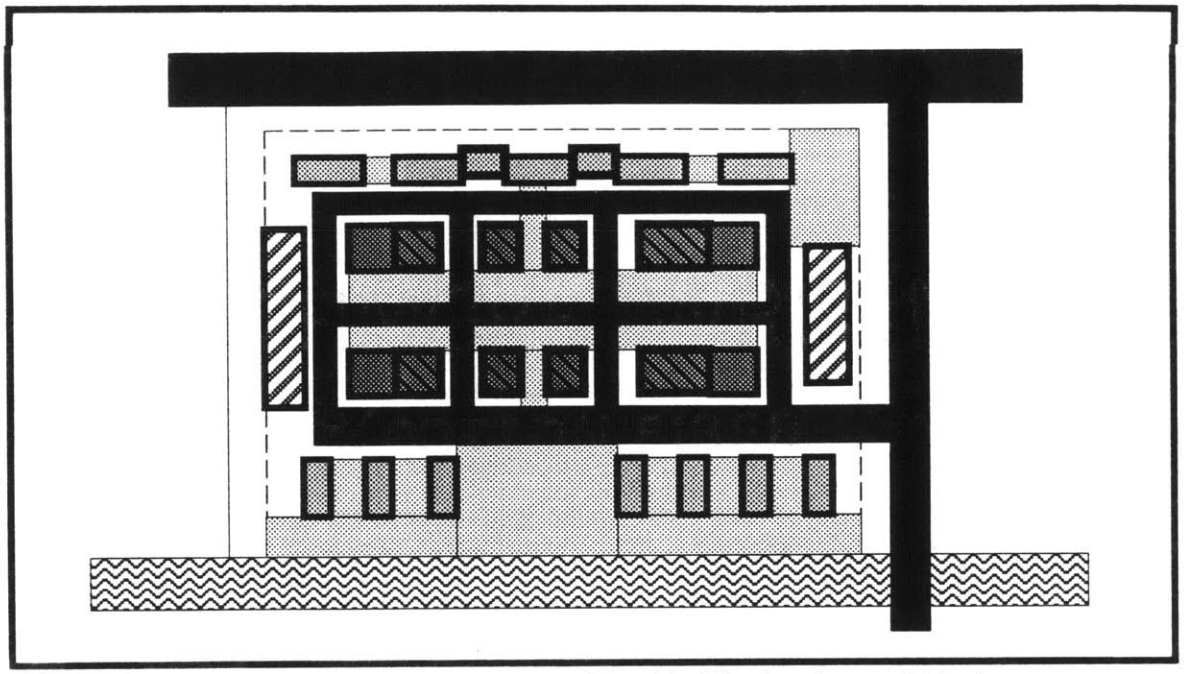
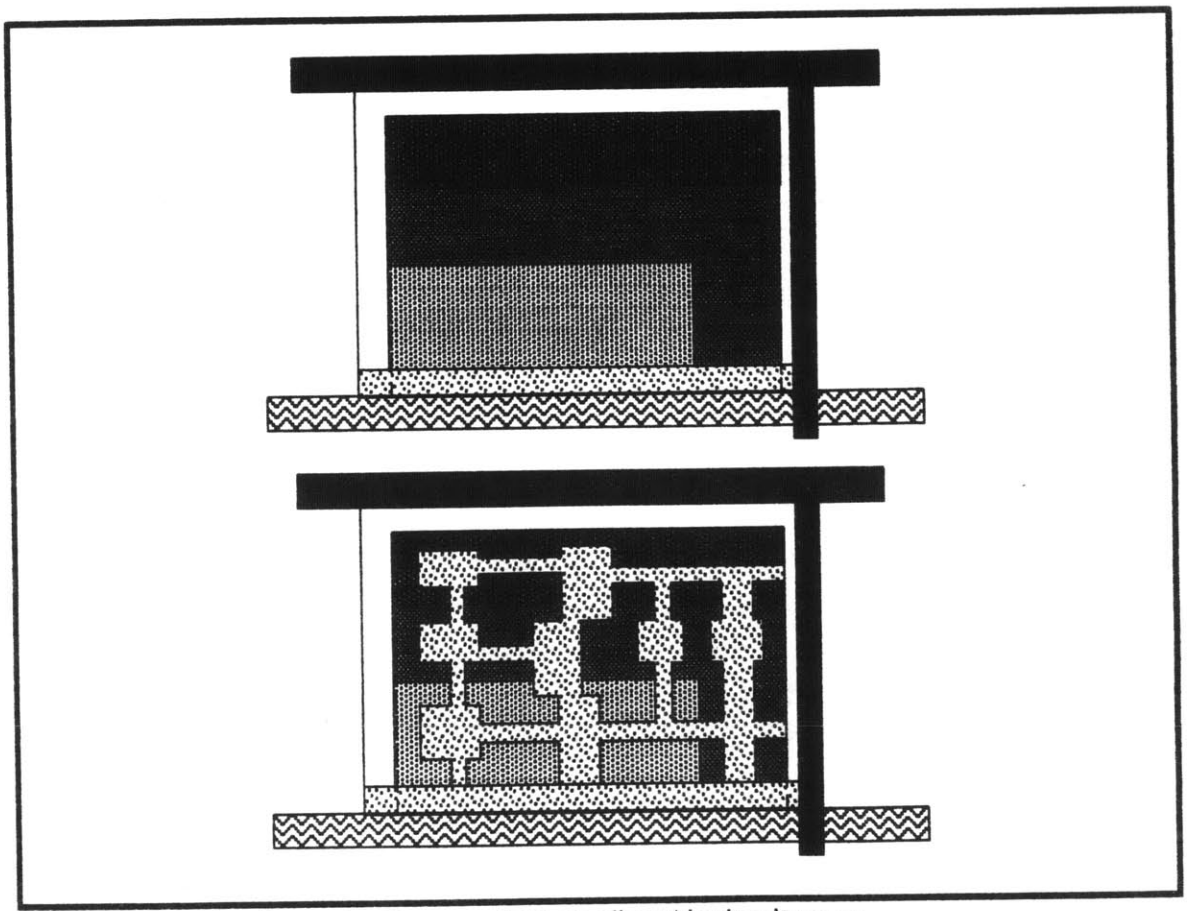


Fig. 4.14 example 2: realized road/objects and fitted building/ and space/objects



Figs. 4.15 & 4.16 "carved-out" spaces from an allowable density map

Flexibility Issues

These examples are meant to show that these modeling tools were not developed with any particular design method or process in mind. A designer who is most interested in the spatial experience and organization of a site, for example, is not forced to first consider the road network. Likewise for the developer who is more concerned with the building program and meeting accessibility and parking requirements; in this case, spatial definition can wait until these issues are addressed. I have intentionally tried not to restrict the order of use or these tools and to allow the free exploration of different priorities.

I have kept this freedom of modeling by keeping the operations on different object types discrete, so that any procedure can be used whenever desired by the designer without significantly affecting the rest of the objects in the design space. This approach limits how much automation is involved, so that most responsibilities and decisions are left to the user. What is required as an end result of designing with these tools and in this environment is that the user eventually find a jigsaw-like fit of the objects in the design, where every part of the site is accounted for and defined. If this is achieved within the program limits, that is in built-area and in functional/access requirements, then the user will have a model that can be further realized and visually tested for spatial and experiential qualities using the 3D and greater design realization procedures.

Potential Users

The stated focus of the uses and functions of these modeling tools has been on the designer, to inform him or her, to allow for experimenting with alternatives, and to communicate and understand the implications of design decisions. But in its current form, these tools would certainly still limit the designer's imagination and function; the included objects and processes and the allowed configurations are just too narrow to allow a designer full creativity. This is something that could be improved dramatically, given time and resources, but the ultimate resolution of these limitations are still in question.

I have often considered the place of urban design in the spectrum of professions; I mentioned earlier that it can be considered a bridge between

architecture and urban planning. Indeed, the tools I have developed could serve a planner well, with some modification of focus. Planners usually do not have much training or experience in considering concepts of morphological design or of *place*. The recommendations and guidelines that planners create are more often solutions to problems of capacity or viability, or of function. The physical ramifications of these guidelines are often beyond the planner's scope, and it is up to the designer to sort out these rules and to create places from them. I feel that a tool for taking quantitative inputs and limits and for easily realizing the possibilities from these inputs would be welcome in the hands of the planner who is concerned with the effects of his plans. With these tools, a planner would not have to *design* as an architect would, but would be able to test rather easily some of the possibilities brought and limitations imposed by his or her recommendations. The idea is that this greater consideration would in the end lead to better planning and planning guidelines.

It has been suggested that the mass modeling and information handling components of this tool would also be of use to real estate developers for capacity analysis. By investigating the physical possibilities (or impossibilities) of desired building programs to meet planning and design guidelines, developers often need to pre-test their proposals to see if any sort of "fit" is at all possible. Usually the developer will hire a design or planning consultant who often has a different or conflicting point of view. By testing for one's self, a developer can learn the benefits and drawbacks of a proposal and in turn explore potentially more satisfying alternatives.

The potential for such a design tool to mediate between various disciplines is an advantage. In my opinion, the computer has the ability to empower individuals with the means to handle problems normally out of their domain of expertise, while still utilizing the individual's own expertise. Such a tool could also be seen as a means of coming together and communicating from different perspectives, with the power to represent multiple points of view in a common format.

Part 5 - In Retrospect:

(Oh, Lord) What Have I Done?

For this thesis, I set out to create some computer-aided design tools that would perform some of the mundane tasks for a designer of large-scale urban environments. My goal was to create some means of automation for the designer, to help him or her avoid the distractions of rote handling of information so that more attention may be given to the design. In the process, I hoped to learn a thing or two about design activity and about working to customize a computer-aided design platform.

To simply this task, I had to make several broad assumptions about the problem and the potential solutions: that the tools would be used only for specific types of design problems, i.e. newly built, district-sized, traditionally urban environments; that the design tools would handle only morphological considerations; that for some types of schematic designing a designer makes best use of modeling tools; etc. Furthermore, the development of the tools for greater design realization presume that it is important to understand urban spatial qualities at the schematic stages of design, and that by incorporating standard types in place of schematic abstractions (if such "standards" even exist) this sort of understanding is possible.

I now realize that what I have accomplished in this study is more than simply writing procedures for task handling; rather, I have created a rudimentary *design environment* on the computer for handling a specific kind of design task.

Unlike the standard, commercially available CAD programs for drawing in 2D and 3D, this environment and the elements in it have *purpose* and *intention.* They exist not merely to indicate graphically the geometric constructs of a design solution, but to participate in the design activity with the designer. It is in their information content and the processes that act upon this information that the intentions they are given are made active. As "tools," what I have created are tools useful only in a certain environment where the "theory" behind their usage can be applied. Much as tools like the screwdriver or the winch apply theories of torque and tension to their use, the tools for cutting and stacking and realizing building/objects apply theories about buildings and their purposes and functions.

The participants in the design environment I have created are the user, the objects of the design (building, space, road, and site), and the procedures or processes for acting upon and transforming those objects. The objects do nothing on their own, and the procedures are dormant until selected; this leaves the user in ultimate control of the environment, bringing his or her full discretion and judgement to bear on the design situation. In a sense, the design environment could be a form of "design world" such as Schoen suggested. I have tried to include a degree of flexibility for the user so that intuition and associative leaps in understanding are still allowed.

The Good, the Bad, and the Inelegant

There are a number of positive results from this project, both in terms of the design environment created and in the method I used to arrive at this environment. My discovery and utilization of object-oriented data and designing is the most pronounced accomplishment I can claim. Of course, others have been using this approach for years, but my understanding of its advantages and my incorporating it into this tool has been a major, unexpected step. I can recognize the power of dealing with "things," or objects, with properties and characteristics, when designing as well as when defining and developing tools for design. To me, creation and invention are more natural when dealing in a world of elements than they are in a world of concepts and numbers. Indeed, the logic of many of the procedures I developed directly

benefited by considering them as physical actions rather than numerical manipulations.

There are a few factors I would consider as drawbacks or failings in this project. The lack of time to realize and explore many of the ideas I had is to be expected of any semester-long thesis project. This shortage of time forced me to simplify many concepts regarding design and urban environments, and to leave out entirely some very relevant and interesting issues. Simplification was also mandated by limitations and constraints of the platform on which this project was based. Although I am quite pleased with MiniCad and MiniPascal in a number of ways, there remains a significant degree of "flakiness" in the software; in the past several weeks I have contributed a great deal to the software development company's "wish list" of improvements for the next release. Many of the procedures will seem to be very roundabout in their trying to get around unexpected (and often unreasonable) limitations of the platform; a great deal of time was expending trying to come up with clever solutions to unnecessarily intractable problems. In short, the limitations of the program I've developed has many limitations, too numerous to mention.

For example, it is very difficult in MiniCad to deal with rectilinear objects, such as buildings tend to be, in a non-rectilinear field or rotated grid. This forced me to consider only design situations within a north-south/east-west framework: certainly not generally applicable. The limitations in 3D editing also forced me to consider modeling only in a 2D environment; acceptable but not preferable. For another example, I later began to realize limitations in the software's capacity for storing object information; the difficulty in accessing and transforming the data types such as strings, numbers and integers required numerous extra procedure steps for interpretation and limiting the sort of data that could be easily dealt with by the procedures. But in retrospect, one of my primary purposes of undertaking this project was to deal with a commercially available CAD product, and to learn to deal with its languages and biases. Working within limits is a fact of life as well as a fact of design.

Lingering Questions

My recognition that what I had developed was more than a set of tools but a design environment came rather late in the project. In hindsight, I could raise a number of questions about the paths I took and how I might have pursued the project differently if this sort of environment had been a goal from the start. Regarding the user-object-process partnership in the environment, there are many alternative ways and degrees that information and responsibility could be delegated among these agents. The decisions I made in this regard were more often dictated by time and platform constraints than by what I felt was best for the designer and the design activity.

One aspect that I might have explored further would be how to incorporate the processes within the objects themselves; the rules and routines held as data. This would be true *object-oriented programming*. If indeed this were possible in MiniCad, the problems involving the development of these routines and determining the object-user partnership, not to mention the impact on the definition of urban design elements, would probably render the project beyond my capabilities. My solution seems to have fallen part-way between the *intelligence* of object-oriented programming for design and inert, geometrical, *dumb* elements of design, but with user in full control.

What would I do next with this, if I had the time and resources? The first things I would want to do would be to improve the procedures, making them less flaky and developing the proposed similar ones. I would also see about getting some rudimentary greater design realizer functioning. After this, I would continue to work on the realizer, developing its library of symbols and improving on its rules of composition. For the modeler and information handler, I would start to work on the site/object that I proposed. The potentials in implementing rules of *value* based on attractiveness and "willingness to pay" for use in the development of real estate could make this tool practically useful. This would require improvements in the object database, so in turn the whole project would gradually move forward. Essentially, all I can see to do with this project is to work on the various proposals I have mentioned throughout this paper. This applies to ideas or areas of study as well, in diagramming, site diagramming, prototypes, application to other scales of development., appropriate visual qualites for schematic realization, educative applications, etc.

If I'd had unlimited flexibility and capability of platform, and a clear sense of purpose to develop a design environment for use on large-scale developments, then the ultimate question is what sort of environment would be best for the designer. The answer in turn would be a simple question: Who specifically is the designer you are referring to? The environment I would develop would be an environment for myself, suited to my tastes and preferences and methods. I don't believe I could expect this particular environment to suit anyone else quite as well as it would suit me. The next user would probably come up with several objections with its organization and assumptions. Therefore, that designer would need to develop a design environment for himself, wouldn't he?

I am a designer, really, who wants to see if he could make a better environment for designing on the computer. Why can't more designers be allowed to do this, and not just the ones with special aptitudes and plenty of time on their hands? I believe that a promising direction for CAD software would be to provide for designers open and definable interfaces and programs for easily encoding one's knowledge regarding the nature of the objects we deal with, as well as rules for dealing with those objects; in a sense, allowing the easy creation of personalized design environments. MiniCad is a small step towards this: it is certainly *definable* for an environment, but not really that *open.* If this is accomplished someday, and I think it may soon be possible given the evolution of programming languages and processing power, then every designer will in a sense be a programmer, encoding his or her knowledge repertoire into a fully personal, intuitive design world of one's own.

Appendix - Code for Procedures Written for this Project in MiniPascal Programming Language

For the benefit of any who may attempt to build upon the work I've done for this thesis, I am providing here the text of the MiniPascal procedures I wrote to accomplish the various tasks I've mentioned. I've tried to be very clear in the thesis text which procedures I was not able to write. Furthermore, of the procedures I did write, there are a few that did not make it into this text by the publication deadline. Let the reader also be warned that many of the procedures included here continue to have major problems, and they almost completely lack any error-trapping routines. Nevertheless, I hope that the publication of these procedures proves helpful and enlightening to the task of programming in MiniPascal.

"PRGRM AREAS"

```
{This procedure prompts the user to input the building program by gross square footage per use.}

PROCEDURE SetAreas;
CONST

{Constant widths for retail and office uses are 50' and for residential is 30'.  Color and pattern
selections are also constants.}
    maxwd = 300;
    retDpth = 50;
    offDpth = 50;
    resDpth = 30;
    FilPat1 = 1;      {single use          all background}
    FilPat2 = 12;     {mix-use     light foreground, heavy background}
    FilPat3 = 37;     {mix-use     equal foreground and background}
    FilPat4 = 14;     {mix-use     heavy foreground, light background}
    retCol = 215;         {bright yellow}
    offCol = 169;         {deep blue}
    resCol = 5;       {bright red}
VAR
    retArea, offArea, resArea :REAL;
    LLorigin : INTEGER;
    SSArea1 : HANDLE;

{This procedure initiates the dialog process for data input}
    PROCEDURE GetNumbers(VAR retArea,offArea,resArea:REAL);
    BEGIN
        retArea := RealDialog('Enter total gross s.f. of retail area:','0'");
        offArea := RealDialog('Enter total gross s.f. of office area:','0'");
        resArea := RealDialog('Enter total gross s.f. of residential area:','0'");
    END;

{This procedure updates the inert baseline of the drawing space to record the builidng program
input}
    PROCEDURE BaseLine(retArea,offArea,resArea:REAL);
    VAR
        h:handle;
        retAreaS, offAreaS, resAreaS:string;
    BEGIN
        h := FObject;
        retAreaS := Num2Str(2,retArea);
        offAreaS := Num2Str(2,offArea);
        resAreaS := Num2Str(2,resArea);
        SetRecord(h,'By use');
        SetRField(h,'By use','retail floors', retAreaS);
        SetRField(h,'By use','office floors', offAreaS);
        SetRField(h,'By use','residential floors', resAreaS);
    END;

{sub procedure for setting record/field values for new b/o's}
    PROCEDURE DefineRecFld(h:handle; type:string);
    VAR fld : string;
    BEGIN
        fld := Concat(type,' floors');
        SetRecord(h,'By Floor');
        SetRField(h,'By Floor','1st flr', type);
        SetRecord(h,'By use');
```

```
        SetRField(h,'By use', fld, '1');
    END;

{sub-procedure draws b/o's in the reserve space}
    PROCEDURE CreateArea(clss:STRING; col,dpth:INTEGER; VAR Darea:REAL);
    LABEL 1;
    VAR last:handle;
    BEGIN
        FillBack(col);
        NameClass('1 story');
        WHILE ((Darea/dpth)>maxwd) DO
            BEGIN
            Rect(LLorigin,0,(LLorigin+dpth),maxwd);
            last := LObject;
            DefineRecFld(last,clss);
            Darea := Darea - (dpth*maxwd);
            LLorigin := LLorigin+80;
            END;
        IF Darea = 0 THEN
        BEGIN
            LLorigin := LLorigin+20;
            GOTO 1;
        END;
        Rect(LLorigin,0,(LLorigin+dpth),(Darea/dpth));
        last := LObject;
        DefineRecFld(last,clss);
        LLorigin := LLorigin+100;
    1:END;

{sub-procedure sets the values for the next set of b/o's to draw}
    PROCEDURE DefineAreas(retArea,offArea,resArea:REAL);
    VAR
        Dcolor, Ddpth : INTEGER;
        Darea : REAL;
        clss : STRING;
    BEGIN
        clss := 'retail';
        Dcolor := retCol;
        Ddpth := retDpth;
        Darea := retArea;
        CreateArea(clss,Dcolor,Ddpth,Darea);
        clss := 'office';
        Dcolor := offCol;
        Ddpth := offDpth;
        Darea := offArea;
        CreateArea(clss,Dcolor,Ddpth,Darea);
        clss := 'residential';
        Dcolor := resCol;
        Ddpth := resDpth;
        Darea := resArea;
        CreateArea(clss,Dcolor,Ddpth,Darea);
    END;

{Main procedure sets initial values, gets program input, sets global constants in the baseline, and
defines new objects to draw.}
BEGIN
    Layer('bldgs');
    PenSize(1);
```

```
    retArea := 0;
    offArea := 0;
    resArea := 0;
    FillPat(1);
    LLorigin := 0;
    GetNumbers(retArea,offArea,resArea);
    BaseLine(retArea,offArea,resArea);
    DefineAreas(retArea,offArea,resArea);
END;
run(SetAreas);
```

"PRGRM PCTS."

{This procedure functions much like the "Prgrm Areas" but with a different dialog input format, and subsequently different formulas for calculating the areas of the b/o's as they are drawn.}

```
PROCEDURE SetAreas2;
CONST
    maxwd = 300;
    retDpth = 60;
    offDpth = 60;
    resDpth = 35;
    FilPat1 = 1;      {single use          all background}
    FilPat2 = 12;     {mix-use     light foreground, heavy background}
    FilPat3 = 37;     {mix-use     equal foreground and background}
    FilPat4 = 14;     {mix-use     heavy foreground, light background}
    retCol = 215;         {bright yellow}
    offCol = 169;         {deep blue}
    resCol = 5;       {bright red}
VAR
    totalSF, retPct, offPct, resPct :  REAL;
    LLorigin : INTEGER;

    PROCEDURE GetNumbers(VAR totalSF, retPct,offPct,resPct:REAL);
    LABEL 1,2;
    VAR answer : BOOLEAN;
    BEGIN
        2:totalSF := RealDialog('Enter the total s.f. of project:','0"");
        IF totalSF <= 0 THEN GOTO 1;
        retPct := RealDialog('Enter percentage of total gross s.f. to be retail area:','0%');
        offPct := RealDialog('Enter percentage of total gross s.f. to be office area:','0%');
        resPct := 100-(retPct+offPct);
        answer := YNDialog('This leaves % for residential.  Do you wish to continue?');
        IF NOT answer THEN GOTO 2;
    1:END;

    PROCEDURE BaseLine(totalSF,retPct,offPct,resPct:REAL);
    VAR
        h:handle;
        retArea, offArea, resArea:real;
        retAreaS, offAreaS, resAreaS:string;
```

```
BEGIN
    h := FObject;
    retArea := (retPct/100)*totalSF;
    offArea := (offPct/100)*totalSF;
    resArea := (resPct/100)*totalSF;
    retAreaS := Num2Str(2,retArea);
    offAreaS := Num2Str(2,offArea);
    resAreaS := Num2Str(2,resArea);
    SetRecord(h,'By use');
    SetRField(h,'By use','retail floors', retAreaS);
    SetRField(h,'By use','office floors', offAreaS);
    SetRField(h,'By use','residential floors', resAreaS);
END;

PROCEDURE DefineRecFld(h:handle; type:string);
VAR fld : string;
BEGIN
    fld := Concat(type,' floors');
    SetRecord(h,'By Floor');
    SetRField(h,'By Floor','1st flr', type);
    SetRecord(h,'By use');
    SetRField(h,'By use', fld, '1');
END;

PROCEDURE CreateArea(clss:STRING; col,dpth:INTEGER; VAR Darea:REAL);
LABEL 1;
VAR last:handle;
BEGIN
    FillBack(col);
    NameClass('1 story');
    WHILE ((Darea/dpth)>maxwd) DO
        BEGIN
        Rect(LLorigin,0,(LLorigin+dpth),maxwd);
        last := LObject;
        DefineRecFld(last,clss);
        Darea := Darea - (dpth*maxwd);
        LLorigin := LLorigin+80;
        END;
    IF Darea = 0 THEN
    BEGIN
        LLorigin := LLorigin+20;
        GOTO 1;
    END;
    Rect(LLorigin,0,(LLorigin+dpth),(Darea/dpth));
    last := LObject;
    DefineRecFld(last,clss);
    LLorigin := LLorigin+100;
1:END;

PROCEDURE DefineAreas(totalSF,retPct,offPct,resPct:REAL);
VAR
    Dcolor, Ddpth : INTEGER;
    Darea : REAL;
    clss : STRING;
BEGIN
    clss := 'retail';
    Dcolor := retCol;
    Ddpth := retDpth;
```

```
        Darea := (retPct/100)*totalSF;
        CreateArea(clss,Dcolor,Ddpth,Darea);
        clss := 'office';
        Dcolor := offCol;
        Ddpth := offDpth;
        Darea := (offPct/100)*totalSF;
        CreateArea(clss,Dcolor,Ddpth,Darea);
        clss := 'residential';
        Dcolor := resCol;
        Ddpth := resDpth;
        Darea := (resPct/100)*totalSF;
        CreateArea(clss,Dcolor,Ddpth,Darea);
    END;

BEGIN
    Layer('bldgs');
    PenSize(1);
    retPct := 0;
    offPct := 0;
    resPct := 0;
    LLorigin := 0;
    GetNumbers(totalSF,retPct,offPct,resPct);
    BaseLine(totalSF,retPct,offPct,resPct);
    DefineAreas(totalSF,retPct,offPct,resPct);
END;
run(SetAreas2);
```

"DEFINE SPACES"

{This is a procedure for drawing space/objects. This allows for creating polygon shapes that are not necessarily rectangular. The sub-procedure for assigning record/field values was not completed when this was printed out.}

```
PROCEDURE spaces;
CONST color=73;
VAR
    x1,y1,x2,y2,x,y:real;
    Count, counter:integer;
    last :handle;

BEGIN
    Layer('spaces');
    Pensize(1);
    SetConstrain('AQWR');
    FillBack(color);
    Message('Draw a polygon around the space.');
    GetPt(x,y);
    MoveTo(x,y);
    x1 := x;
    y1 := y;
{points are selected in the design space until the first point is selected again.}
```

```
    REPEAT
        GetPtL(x1,y1,x2,y2);
        LineTo(x2,y2);
        Count := Count+1;
        x1 := x2;
        y1 := y2;
    UNTIL (x2=x) AND (y2=y);
    ClrMessage;
    DoMenu(MCombineSf,NoKey);
    DeleteObjs;
    ClrMessage;
    SetConstrain('AQ');
    SetTool(2);
END;
RUN(spaces);
```

"STREETS & ROADS"

{This procedure uses dialog boxes for requesting road width information. The user is then prompted to pick begin- and end-points for a road segment, and is then asked if s/he would like to continue.}

```
PROCEDURE drawroad;
CONST lnwdth = 10;
VAR
    lanes, ppark, wdth:integer;
    lanesS, pparkS, wdthS:string;
    x1,y1,x2,y2:real;
    road:handle;
    answer:boolean;

    PROCEDURE MAKEROADS;
    BEGIN
        DSelectAll;
        layer('roads');
        lanes := IntDialog('How many traffic lanes wide to you want this road segment?',lanesS);
        ppark := IntDialog('Do you want parallel parking on both sides [2], one side [1], or none
[0]?',pparkS);
        wdth := (lanes+ppark)*lnwdth;
        lanesS := Num2StrF(lanes);
        pparkS := Num2StrF(ppark);
        wdthS := Num2StrF(wdth);
        PenSize(lanes*20);
        Message('Pick begin and end points of road segments.');
        GetPt(x1,y1);
        GetPtL(x1,y1,x2,y2);
        MoveTo(x1,y1);
        LineTo(x2,y2);
        road := LSActLayer;
        SetClass(road,'street');
        SetRecord(road,'street');
```

```
        SetRField(road,'street','lanes', lanesS);
        SetRField(road,'street','parallel parking',pparkS);
        SetRField(road,'street','width', wdthS);
        answer := YNDialog('Would you like to continue creating road segments?')
    END;

BEGIN
    answer := TRUE;
    lanesS := '4';
    pparkS := '0';
    SetConstrain('AQW');
    REPEAT
        makeRoads;
    UNTIL NOT answer;
    ClrMessage;
    PenSize(1);
    SetConstrain('AQ');
END;
Run(drawroad);
```

"PARKING AREAS"

{This procedure allows creation of parking areas much like "Define Spaces" above, but asks the user if the area is to be a structure, and if so, how many levels. Also does not yet have sub-procedures for creating record/field data.}

```
PROCEDURE parking;
CONST color=248;
VAR
    x1,y1,x2,y2,x,y:real;
    Count, counter:integer;
    last, pkarea :handle;
    answer :boolean;

    PROCEDURE structure;
    VAR
        Levels, newLW:integer;
    BEGIN
        Levels := IntDialog('How many levels [above ground]?','3');
        newLW := Levels * 17;
        SetLW(pkarea,newLW);
    END;

BEGIN
    Layer('spaces');
    Pensize(1);
    SetConstrain('AQWR');
    FillBack(color);
    Message('Draw a polygon around the space.');
    GetPt(x,y);
    MoveTo(x,y);
```

```
    x1 := x;
    y1 := y;
    REPEAT
        GetPtL(x1,y1,x2,y2);
        LineTo(x2,y2);
        Count := Count+1;
        x1 := x2;
        y1 := y2;
    UNTIL (x2=x) AND (y2=y);
    ClrMessage;
    DoMenu(MCombineSf,NoKey);
    DeleteObjs;
    pkarea := LActLayer;
    answer := YNDialog('Do you want this object to be a parking structure?');
    IF answer THEN structure;
    ClrMessage;
    SetConstrain('AQ');
    SetTool(2);
END;
RUN(parking);
```

"ROADS TO BLOCKS"

{This procedure was a precursor to "Real Roads", and is a different form of greater realization of road/objects. Instead, this goes a bit further to create the blocks as objects, deleting the realized road/objects. "Real Roads" adapted and simplified this routine.}

```
PROCEDURE roads_blocks;
VAR
    base :handle;


{sub-procedure for making the subsurface under the block/objects, revealed a road surfaces.}
    PROCEDURE makeSubStrat;
    BEGIN
        Layer('backgrnd');
        DSelectAll;
        base := FActLayer;
        SetSelect(base);
        DoMenu(MDuplicate,NoKey);
        base := LSActLayer;
        SetFillBack(base,249);
        SetDSelect(base);
    END;


{This actually creates the realized road/objects and was subsequently used in "Real Roads"}
    PROCEDURE SubRoads   (h:handle);
    VAR
        x1,y1,x2,y2,lnAngle,wdt,ppark,lng,ang1,ang2:real;
    BEGIN
```

```
        Layer('backgrnd');
        FillBack(250);
        AngleVar;
        Pensize(1);
        wdt := Eval(h,'street'.'width');
        ppark := Eval(h,'street'.'parallel parking');
        GetSegPt1(h,x1,y1);
        GetSegPt2(h,x2,y2);
        lnAngle := HAngle(h);
        wdt := wdt/2;
        lng := Distance(x1, y1,x2, y2);
        ang1 := lnAngle + 90;
        ang2 := lnAngle -90;
        MoveTo(x1,y1);
        Relative;
        Poly(0,0,wdt,#ang1,lng,#lnAngle,(2*wdt),#ang2, lng, #(lnAngle+180));
    END;


{Next few subprocedures are for clipping the roads/objects out and leaving the block/objects.}
    PROCEDURE ClipOut(h:handle);
    BEGIN
        SetSelect(h);
        DoMenu(MClipSurf,NoKey);
        SetDSelect(h);
    END;

    PROCEDURE PickIt(h:handle);
    BEGIN
        SetSelect(h);
    END;

    PROCEDURE clipBlocks;
    VAR fld:handle;
    BEGIN
        Layer('backgrnd');
        DSelectAll;
        fld := FActLayer;
        SetSelect(fld);
        ForEachObject(ClipOut, ((L='backgrnd') & (FB=250)));
        DSelectAll;
        ForEachObject(PickIt,(FB=250));
        DeleteObjs;
    END;

BEGIN
    ClosePoly;
    makeSubStrat;
    ForEachObject(subRoads,L='roads');
    clipBlocks;
    SetSelect(base);
    MoveBack;
END;
RUN(roads_blocks);
```

"CLIP OFF"

{This was a particularly tricky routine for clipping segments out of a building object, leaving the same data in all the new created objects. Required a very complex substitution process for retaining the data, which would then be read from the temporary holding objects to the newly created objects. Very weird, if you ask me. Also very flaky and unreliable; needs a lot of error trapping, which I did not have time to do.}

```
PROCEDURE ClipIt;
VAR
    x1,y1,x2,y2 : REAL;
    clipper, pieces : HANDLE;
    Countr, LCount1, LCount2, LCountDif : LONGINT;
    aCode : INTEGER;

{sub-procedure prompts user to draw a rectangle to clip out}
    PROCEDURE ClipBox(VAR h:handle);
    BEGIN
        Message('Draw a box around the selected area to clip.');
        GetRect(x1,y1,x2,y2);
        Rect(x1,y1,x2,y2);
        h := LObject;
    END;

{transfers object data from an object "below" to an object "on top" of it}
    PROCEDURE ReadBelow(h:handle);
    VAR
        below : handle;
        x,y : real;
        clss, rcrd, fl1, fl2, fl3, fl4, fl5, ret, off, res : string;
    BEGIN
        HCenter(h,x,y);
        below := PickObject(x,y);
        SetSelect(below);
        clss := GetClass(below);
        fl1 := EvalStr(below, ('By Floor'.'1st flr'));
        fl2 := EvalStr(below, ('By Floor'.'2nd flr'));
        fl3 := EvalStr(below, ('By Floor'.'3rd flr'));
        fl4 := EvalStr(below, ('By Floor'.'4th flr'));
        fl5 := EvalStr(below, ('By Floor'.'5th flr'));
        ret := EvalStr(below, ('By use'.'retail floors'));
        off := EvalStr(below, ('By use'.'office floors'));
        res := EvalStr(below, ('By use'.'residential floors'));
        {retS := Num2StrF(ret);
        offS := Num2StrF(off);
        resS := Num2StrF(res);}
        SetClass(h, clss);
        SetRecord(h,'By Floor');
        SetRecord(h,'By use');
        SetRField(h, 'By Floor', '1st flr', fl1);
        SetRField(h, 'By Floor', '2nd flr', fl2);
        SetRField(h, 'By Floor', '3rd flr', fl3);
        SetRField(h, 'By Floor', '4th flr', fl4);
        SetRField(h, 'By Floor', '5th flr', fl5);
        SetRField(h, 'By use', 'retail floors', ret);
        SetRField(h, 'By use', 'office floors', off);
```

```
        SetRField(h, 'By use', 'residential floors', res);
    END;

    PROCEDURE AssignData;
    BEGIN
        pieces := FActLayer;
        FOR Countr := 1 TO LCountDif DO
        BEGIN
            ReadBelow(pieces);
            pieces := NextObj(pieces);
        END;
    END;

BEGIN
    {DoMenu(MPref,NoKey);}
    Layer('bldgs');
{tallies the number of objects clipped from}
    LCount1 := Count(ALL);
    DoMenu(MDuplicate,NoKey);
    LCount2 := Count(ALL);
    LCountDif := LCount2 - LCount1;
    Redraw;
    ClipBox(clipper);
{MIntSurf creates new clippings but does not subtract from the original objects}
    DoMenu(MIntSurf,NoKey);
{changes which objects are selected for use in ClipSurf}
    FOR Countr := 1 TO LCountDif DO
    BEGIN
        pieces := LSActLayer;
        SetDSelect(pieces);
    END;
{MClipSurf does the subtracting from the original pieces that MIntSurf did not do}
    DoMenu(MClipSurf,NoKey);
{all this changes again which objects are currently selected}
    SetDSelect(clipper);
    MoveBack;
    DSelectAll;
    SetSelect(clipper);
    DeleteObjs;
{AssignData  is a sub-procedure shown above}
    AssignData;
{once again, reselecting the objects}
    DSelectAll;
    pieces := LObject;
    FOR Countr := 1 TO LCountDif DO
    BEGIN
        SetSelect(pieces);
        pieces := PrevObj(pieces);
    END;
    MoveBack;
    DSelectAll;
    AssignData;
{deletes all objects created for just holding data}
    DeleteObjs;
    pieces := FActLayer;
    FOR Countr := 1 TO LCountDif DO
    BEGIN
        SetSelect(pieces);
```

```
        pieces := NextObj(pieces);
    END;
    MoveObjs(10,50,true,false);
    ClrMessage;
END;
RUN(ClipIt);
```

"STACK ON"

{This procedure is for creating a complex "multi-story" object from two "shorter" objects, updating the record/field values and the graphic attributes to reflect this event. Writing this procedure was when I discovered MiniPascals limitations regarding passing arrays a parameters to subprocedures. This program is therefore much longer than it would have been otherwise.}

```
PROCEDURE stack1;
CONST
    Pat0pct = 1;
    Pat20pct = 12;
    Pat40pct = 50;
    Pat60pct = 51;
    Pat80pct = 57;
VAR
    stkOn, stkTo : handle;
    newPat, newLW, NoRet, NoOff, NoRes : integer;
    stkOnArea, stkToArea, residArea : real;
    onList, toList, newList:ARRAY [1..9] of string;

{reads the object under a picked point}
    PROCEDURE GetPiece(VAR stkOb : handle);
    VAR x,y : real;
    BEGIN
        GetPt(x,y);
        stkOb := PickObject(x,y);
    END;

{reads the r/f values of selected objects into arrays}
    PROCEDURE ReadData;
    BEGIN
        ToList[1] := EvalStr(stkto,('By Floor'.'1st flr'));
        OnList[1] := EvalStr(stkon,('By Floor'.'1st flr'));
        ToList[2] := EvalStr(stkto,('By Floor'.'2nd flr'));
        OnList[2] := EvalStr(stkon,('By Floor'.'2nd flr'));
        ToList[3] := EvalStr(stkto,('By Floor'.'3rd flr'));
        OnList[3] := EvalStr(stkon,('By Floor'.'3rd flr'));
        ToList[4] := EvalStr(stkto,('By Floor'.'4th flr'));
        OnList[4] := EvalStr(stkon,('By Floor'.'4th flr'));
        ToList[5] := EvalStr(stkto,('By Floor'.'5th flr'));
        OnList[5] := EvalStr(stkon,('By Floor'.'5th flr'));
        ToList[6] := EvalStr(stkto,('By use'.'retail floors'));
        OnList[6] := EvalStr(stkon,('By use'.'retail floors'));
        ToList[7] := EvalStr(stkto,('By use'.'office floors'));
```

```
        OnList[7] := EvalStr(stkon,('By use'.'office floors'));
        ToList[8] := EvalStr(stkto,('By use'.'residential floors'));
        OnList[8] := EvalStr(stkon,('By use'.'residential floors'));
    END;

{counts the number of floors of the two objects}
    PROCEDURE NoFlrsOn(VAR numFlrs:integer);
    VAR
        count:integer;
        flr : string;
    BEGIN
        count := 1;
        flr := OnList[count];
        WHILE flr<>'' DO
        BEGIN
            count := count+1;
            flr := OnList[count];
        END;
        numFlrs := count-1;
    END;

    PROCEDURE NoFlrsTo(VAR numFlrs:integer);
    VAR
        count:integer;
        flr : string;
    BEGIN
        count := 1;
        flr := toList[count];
        WHILE flr<>'' DO
        BEGIN
            count := count+1;
            flr := toList[count];
        END;
        numFlrs := count-1;
    END;

{If combined objects total more than 5 floors}
    PROCEDURE ExtraFloors(i:integer);
    VAR answer:boolean;
    BEGIN
        answer := YNDialog('Stack is more than 5 floors; extra top floors will go to residual.  Do you
want to continue?');
        IF answer THEN
        BEGIN
            Message('need to work on residual collection procedures');
        END;
    END;

{Tallies values for the new object}
    FUNCTION CountByUse(use:string):integer;
    VAR
        countList, countUse:integer;
        flrUse, countUseS:string;
    BEGIN
        countUse := 0;
        FOR countList := 1 TO 8 DO
        BEGIN
            flrUse := NewList[countList];
```

```
            IF flrUse = use THEN countUse:= countUse+1;
        END;
    CountByUse := countUse;
    END;

{Creates an array for the new object}
    PROCEDURE NewData;
    VAR
        onflrs, toflrs, newflrs, byUse, count:integer;
        byUseS, newFlrsS:string;
    BEGIN
        NoFlrsOn(onflrs);
        NoFlrsTo(toflrs);
        newflrs := onflrs + toflrs;
        IF newflrs>5 THEN
        BEGIN
            extrafloors(newflrs);
            newflrs := 5;
        END;
        newFlrsS := Num2StrF(newflrs);
        NewList[9] := newFlrsS;
        FOR count := 1 TO toflrs DO
            NewList[count] := ToList[count];
        FOR count := (toflrs+1) TO newflrs DO
            NewList[count] := OnList[count-toflrs];
        byUse := CountByUse('retail');
        byUseS := Num2StrF(byUse);
        NewList[6] := byUseS;
        byUse := CountByUse('office');
        byUseS := Num2StrF(byUse);
        NewList[7] := byUseS;
        byUse := CountByUse('residential');
        byUseS := Num2StrF(byUse);
        NewList[8] := byUseS;
    END;

{Determines proper graphic characteristics for new object}
    PROCEDURE FixPat;
    VAR
        flr1Use, TotFlrsS:string;
        iter, count:integer;
        fract, TotFlrs:real;
    BEGIN
        flr1Use := NewList[1];
        FOR iter := 1 TO 5 DO
        BEGIN
            IF NewList[iter]=flr1Use THEN count := count + 1;
        END;
        TotFlrsS := NewList[9];
        TotFlrs := Str2Num(TotFlrsS);
        fract := (count/TotFlrs) * 100;
        IF fract = 0 THEN NewPat := Pat0pct
        ELSE
            IF fract <= 30 THEN NewPat := Pat20pct
            ELSE
                IF fract <= 50 THEN NewPat := Pat40Pct
                ELSE
                    IF fract <= 70 THEN NewPat := Pat60pct
```

```
                    ELSE NewPat := Pat80pct;
    END;

    PROCEDURE FixStories;
    VAR
        stories:real;
        storiesS:string;
    BEGIN
        storiesS := NewList[9];
        stories := Str2Num(storiesS);
        newLW := stories * 17;
    END;

    PROCEDURE FixArea;
    VAR
        stkOnArea, stkToArea : real;
    BEGIN
        stkOnArea := HArea(stkOn);
        stkToArea := HArea(stkTo);
        residArea := stkOnArea - stkToArea;
    END;

{sends message to user about square footage differences}
    PROCEDURE drawResid;
    BEGIN
        IF (residArea > 0) THEN AlrtDialog('Adding excess area to Residual collection');
        IF (residArea < 0) THEN AlrtDialog('Removing area difference from Residual collection');
    Message('Residual area equals: ', residArea:6:2);
    END;

{as it says, creates the new object}
    PROCEDURE CreateNew;
    VAR
        topColor,botColor:real;
        clss:string;
    BEGIN
        topColor := Eval(stkOn,(FB));
        botColor := Eval(stkTo,(FB));
        SetFillFore(stkTo, topColor);
        SetFillBack(stkTo, botColor);
        SetFPat(stkTo, newPat);
        SetLW(stkTo, newLW);
        SetRField(stkTo,'By Floor','1st flr',NewList[1]);
        SetRField(stkTo,'By Floor','2nd flr',NewList[2]);
        SetRField(stkTo,'By Floor','3rd flr',NewList[3]);
        SetRField(stkTo,'By Floor','4th flr',NewList[4]);
        SetRField(stkTo,'By Floor','5th flr',NewList[5]);
        SetRField(stkTo,'By use','retail floors',NewList[6]);
        SetRField(stkTo,'By use','office floors',NewList[7]);
        SetRField(stkTo,'By use','residential floors',NewList[8]);
        clss := Copy(NewList[9],1,1);
        clss := Concat(clss, ' stories');
        SetClass(stkTo,clss);
    END;

{gets rid of the old objects}
    PROCEDURE RidStkOn;
    BEGIN
```

```
        DSelectAll;
        SetSelect(stkOn);
        DeleteObjs;
    END;

BEGIN
    Layer('bldgs');
    Message('Select object to be stacked upon.');
    GetPiece(stkTo);
    Message('Select object to stack onto previously selected object.');
    GetPiece(stkOn);
    ReadData;
    NewData;
    FixPat;
    FixStories;
    FixArea;
    DrawResid;
    CreateNew;
    RidStkOn;
END;
RUN(stack1);
```

"AREA?"

{very simple procedure for calculating the total area of selected building objects}

```
PROCEDURE whatarea;
VAR
    Xarea:LONGINT;
    h:handle;

BEGIN
    Xarea := 0;
    h := FSActLayer;
    WHILE h <> NIL DO
        BEGIN
        Xarea := Xarea + HArea(h);
        h := NextSObj(h);
        END;
    Message('These objects have an area of ', Xarea, ' s.f.');
END;
RUN(whatarea);
```

"UPDATE AREAS"

{This procedure is the primary information handling component. Calculates the total area in the work space, checks for difference with the globally held building program values, and add or subtracts difference from objects in the reserve space. Needs work still.}

```
PROCEDURE updateit;
CONST ht = 300;
VAR
    PRet, POff, PRes, ARet,AOff,ARes,DRet,DOff,DRes:real;

{Reads globally held values}
    PROCEDURE GetProgram;
    VAR
        h:handle;
        StrVal:string;
    BEGIN
        Layer('0');
        h := FActLayer;
        PRet := Eval(h,'By use'.'retail floors');
        POff := Eval(h,'By use'.'office floors');
        PRes := Eval(h,'By use'.'residential floors');
        Layer('bldgs');
    END;


{Calculates total areas in the work spaces}
    PROCEDURE GetActual;
    BEGIN
        ARet := (AREA(('By use'.'retail floors'=1))) + ((AREA(('By use'.'retail floors'=2)))*2) +
((AREA(('By use'.'retail floors'=3)))*3) + ((AREA(('By use'.'retail floors'=4)))*4) + ((AREA(('By
use'.'retail floors'=5)))*5);

        AOff := (AREA(('By use'.'office floors'=1))) + ((AREA(('By use'.'office floors'=2)))*2) +
((AREA(('By use'.'office floors'=3)))*3) + ((AREA(('By use'.'office floors'=4)))*4) + ((AREA(('By
use'.'office floors'=5)))*5);

        ARes := (AREA(('By use'.'residential floors'=1))) + ((AREA(('By use'.'residential
floors'=2)))*2) + ((AREA(('By use'.'residential floors'=3)))*3) + ((AREA(('By use'.'residential
floors'=4)))*4) + ((AREA(('By use'.'residential floors'=5)))*5);
    END;


{changes the size of objects in the reserve space}
    PROCEDURE AdjResid(Diff:real;use:string);
    LABEL 1;
    VAR
        h,new:handle;
        hUse,fld:string;
        x1,y1,x2,y2,color,wdth,addHght,hHght:real;
    BEGIN
        h := LSActLayer;
        WHILE h<>NIL DO
        BEGIN
            hUse := EvalStr(h,'By Floor'.'1st flr');
            hHght := HHeight(h);
            IF ((hUse=use) and (hHght<300)) THEN
                BEGIN
```

```
                    color := Eval(h,(FB));
                    GetBBox(h,x1,y1,x2,y2);
                    SetName(h,'goner');
                    DelName('goner');
                    wdth := Distance(x1,0,x2,0);
                    addHght := Diff/wdth;
                    x1 := x1 + addHght;
                    FillBack(color);
                    Rect(x1,y1,x2,y2);
                    new := LObject;
                    SetRecord(new,'By Floor');
                    SetRecord(new,'By use');
                    SetRField(new,'By Floor','1st flr', use);
                    fld := Concat(use,' floors');
                    SetRField(new,'By use',fld,'1');
                    GOTO 1;
                END
            ELSE
            h := PrevSObj(h);
        END;
    message('You have overdrawn your account, sir!');
    1:END;

    PROCEDURE FixResid;
    VAR
        count:integer;
        h:handle;
    BEGIN
        Layer('bldgs');
        DSelectAll;
        FOR count := 0 TO 50 DO
            BEGIN
                h := PickObject(count*30,0);
                SetSelect(h);
            END;
        DRes := PRes - ARes;
        DOff := POff - AOff;
        DRet := PRet - ARet;
        IF DRes <> 0 THEN AdjResid(DRes,'retail');
        IF DOff <> 0 THEN AdjResid(DOff,'office');
        IF DRet <> 0 THEN AdjResid(DRet,'residential');
    END;

BEGIN
    GetProgram;
    GetActual;
    FixResid;
END;
RUN(updateit);
```

{I would like to rewrite this to delete the reserve objects first, then calculate the total areas in the work space, find the difference, and using sub-procedures from "Prgrm Areas" re-create a whole new set in the reserve space. Much more straightforward than this method.}

"3D EXTRUDE"

```
{This procedure creates a 3D realization of the 2D schematic model}

PROCEDURE extrusions;

{Subprocedure reads each b/o and extrudes it to the height as indicated by the "class"}
    PROCEDURE bldgExtrude(h:handle);
    LABEL 1;
    CONST
        flrht = 11;
    VAR
        flrs,ht,Height, Width, Depth:real;
        flrsS:string;
    BEGIN
        flrsS := GetClass(h);
        flrsS := Copy(flrsS,1,1);
        IF ((flrsS<'1') OR (flrsS>'5')) THEN GOTO 1;
        flrs := Str2Num(flrsS);
        ht := flrs * flrht;
        SetSelect(h);
        DoMenu(MDuplicate,NoKey);
        DoMenu(MExtrude,NoKey);
        h := LSActLayer;
        Get3DInfo(h,Height, Width, Depth);
        Set3DInfo(h,(Height+335),(Width-35), ht);
        DSelectAll;
    1:END;

{adding upon the "Roads to Blocks" routine, extrudes the block objects 1' to form curbs}
    PROCEDURE blockExtrude(h:handle);
    CONST
        curbht = 1;
    VAR
        Height, Width, Depth:real;
    BEGIN
        SetSelect(h);
        SetDSelect(h);
        DoMenu(MExtrude,NoKey);
        h := LActLayer;
        Get3DInfo(h,Height, Width, Depth);
        Set3DInfo(h,(Height+335),(Width-35),curbht);
        DSelectAll;
    END;

    PROCEDURE MakeSubsurf3D;
    VAR
        h:handle;
        Height, Width, Depth:real;
    BEGIN
        DSelectAll;
        Layer('backgrnd');
        h := FActLayer;
        SetSelect(h);
        DoMenu(MDuplicate,NoKey);
        Layer('3D Model');
```

```
        DoMenu(MExtrude,NoKey);
        h := LActLayer;
        Get3DInfo(h,Height, Width, Depth);
        message('depth = ',depth:5:2);
        Set3DInfo(h,(Height+335),(Width-35),0.01);
        DSelectAll;
    END;

BEGIN
    Layer('3D Model');
    ForEachObject(bldgExtrude,(L='bldgs'));
    ForEachObject(blockExtrude,((L='backgrnd') and (FB=246)));
    MakeSubsurf3D;
    Redraw;

END;
```

Bibliography

Abelson, Harold and Sussman. Structure and Interpretation of Computer Programs. Cambridge: MIT Press, 1985.

Alexander, Christopher. A Pattern Language: Towns, Buildings, Construction. New York: Oxford University Press, 1977.

Alexander, Christopher. Notes on the Synthesis of Form, Cambridge:Harvard University Press, 1964.

Arnheim, Rudolph. Visual Thinking, Berkeley: University of California Press, 1969.

Bender, "The Question of Style in Research."

Calthorpe Associates. "Transit-Oriented Development Guidelines", Sacaramento County Planning & Development Department, 1990.

Crosley, Mark L. "Design Technology: The Next Wave", Progressive Architecture. October 1991, pp. 113-5.

Ervin. "Computer-Aided Diagramming and the 'Generator-Test' Cycle", MIT.

Fargas, Josep. "Designing Rules: Heuristics of Invention in Design", MCP and SMArchS Thesis, MIT Department of Urban Studies and Planning, 1991.

Fleisher, Aaron. "Grammatical Architecture?" 1989.

Goodman, Nelson. Ways of Worldmaking, Indianapolis: Hackett Publishing, 1978.

Gross, Mark. "Design as the Exploration of Constraints", PhD Dissertation, MIT Department of Architecture, 1985.

Habraken, John. The Appearance of the Form, Cambridge: Atwater Press, 1985.

Hillier, Bill and J. Hanson, The Social Logic of Space. Cambridge: Cambridge University Press, 1984.

Hillier, "Spatial Configuration and Use Density at the Urban Level."

Kalay, Y. (Ed.), Computability of Design, New York: Wiley and Sons, 1987.

Krier, Leon. "Houses, Palaces, Cities," Architectural Design, vol. 54, 8 July 1984.

Larson, Kent. "Computer Modeling as a Design Tool", Progressive Architecture. October 1991, pp. 117-9.

Leestma, S. and Nyhoff, L. Pascal Programming and Problem Solving, New York: Macmillan Publishing.

Lischewski, Hans-Christian. "Advanced Techniques in CAD Management," Progressive Archiecture. October 1991, pp. 123-5.

MinCad+ 3.1 User's Guide, Diehl Graphsoft, Inc., 1990.

Minsky, Marvin. Society of Mind, New York: Simon & Schuster, 1986.

Mitchell, William. Computer-Aided Architectural Design, New York: Petrocelli/ Charter, 1977.

Negroponte, Nicholas. Reflections on Computer Aids to Design and Architecture, New York: Petrocelli/Charter, 1975.

Pohl, J, Myers, et al. "A Computer-Based Design Environment: Implemented and Planned Extensions of the ICADS Model", Design Institute Report. CADRU-06-92, California Polytechnic Stat University, Janurary 1992.

Schon, "Designing: Rules, Types and Worlds", Design Studies. Volume 9, Number 3, 1988.

Schon, Educating the Reflective Practitioner. San Francisco: Jossey-Bass, 1987.

Webber, Melvin. "The Urban Place and the Nonplace Urban Realm," Explorations into Urban Structure, Philadelphia: University of Pennsylvania Press, 1964.