The Conceptual Design of Architectural Form: A Performance Spec for a Computer System

by

Peter Harold Jurgensen

Bachelor of Architecture
Illinois Institute of Technology
1982

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 1986

Copyright (c) 1986 Peter Harold Jurgensen

Signature of Author [illegible]
Peter Harold Jurgensen
Department of Architecture
May 16, 1986

Certified by [illegible]
William L. Porter
Professor of Architecture and Planning
Thesis Supervisor

Accepted by [illegible]
Julian Beinart
Chairman, Departmental Committee for Graduate Students

MASS. INST. TECH.
JUN 0 4 1985
LIBRARIES

Rotch

The Conceptual Design of Architectural Form: A Performance Spec for a Computer System

by

Peter Harold Jurgensen

Submitted to the Department of Architecture on May 16, 1986 in partial fulfillment of the requirements for the degree of Master of Science in Architecture Studies.

Abstract

Design is not considered as a professional activity that entails preliminary design, schematic design, and design development, but rather as a creative and conceptual activity that involves the graphic expression of ideas. By using computer graphics, the computer as a protean machine provides the means to develop an instrument for design that is more expressive than the pencil.

Computer graphics systems for architects are built almost exclusively for drafting and working drawing production. Most are based on systems designed for mechanical engineering of machine parts. The drafting paradigm of the CAD system implies that the drawings done on computer represent schemes in an already designed form. The antithetical device is called a creatrix.

The creatrix is analogous to three instruments. It expresses the architect's ideas like a musical instrument. It manages and manipulates building complexity like mathematical instruments. It quantifies and qualifies for the architect's evaluation like scientific instruments.

Its three-fold mandate is: to assist the architect's process of visualization through his expression of ideas; to manage building complexity in a way that allows the architect to think about architecture; to re-present the architect's ideas in a way that helps the architect to critically and objectively evaluate his concepts.

This thesis will present a specification for a system that will satisfy the mandates. The creatrix will be specified in the context of what is feasible technologically now at a reasonable cost.

Thesis Supervisor:	William L. Porter
Title:	Professor of Architecture and Planning

Acknowledgments

The ideas that can be communicated depend on the primitives that can be represented. Words are poor at representing my appreciation of my friends, and make expressions of gratitude seem superficial. We work with what we have.

Thank you Janet Kjerne and Marie Volkers, my much loved sisters.

Thanks to the people that have shared their thoughts, and helped me to be clearer with mine. Especially Doug Stoker, my reader.

Thanks to all the people that have been at MIT that were involved with computers and design in the early days--they blazed the trails. Especially to Bill Porter, who doubled as my advisor.

This thesis is dedicated to my much loved parents.

Contents

Chapter 1

Introduction

Luddites will object. Why would anyone want to use a computer to design with, isn't it artificial? How does one justify computerizing something that seems to work just fine the way it is? There are a number of reasons commonly given: same results faster; higher quality results; more complex situations handled; increase in productivity; delegation of menial tasks; management of more data; reduction of errors... All of the reasons can all be boiled down to two: reduction in time and reduction in labor. As for all of technology, these reasons only show half of double edged blades. They may reduce the number of errors, but they may make errors more serious or easier to make. The real question that needs to be asked of a computer system is: does it de-skill or enhance? It will change work habits. Head designers will be doing more of their own drawings, draftsmen will be producing more presentation drawings. I believe that the computer is capable of enhancing work. I hope to help it do that.

I have no doubt that computers could soon be able to completely synthesize architectural plans, given a number of specifications (such as the spaces needed and their functions, interrelationships, and so on), using various techniques such as expert systems, constraint languages, and heuristically guided search. Technically, these buildings might be very good, but they will have all the quality of elevator music.

A less extreme use of computers in architectural design will have the machine make suggestions and evaluations of proposed schemes. A sort of design partner that watches over your shoulder. An architect that uses this kind of computer system will produce buildings that will have all the quality of a junior high school band.

Both of these uses have their valid places in design, and will become common and necessary in areas like electronic chip design, even in some technical aspects of architectural design. These artificial intelligence systems will be very impressive. My goal is much more humble. I just want to deal with the creative aspects of design, the things that paper and pencil are used for. I am happy to let the designer synthesize, evaluate, and make decisions about a scheme manually; in fact, I insist on it because creative design is an important human activity. I want to develop an instrument (called a creatrix) that will manage a description of a project in a way that will help the designer make better decisions. The creatrix should not be an intellectual prosthetic device.

Our culture equates drawing ability with design ability. This is due to an oversimplification of the relationship between a designer's drawing ability and his design ability. Having good ideas, and expressing them, are two very different things. A creatrix will help express them, which in turn will help produce them.

It may be true that someone facile with colors, materials, proportion, texture, etc. in design should also be able to express those skills in a drawing. But it is not true that someone who can draw a beautiful image of a building will necessarily be good at creating buildings with meaningful spaces, built with appropriate materials, beautiful proportions...

The next chapter will discuss important events in the history of computers as applied to design, what was good and bad. The third will consider representations, drawings, and models, and how they encode information. The fourth will describe design as an activity, this will help define how a computer can be used in design. The next gives an idea of what a creatrix will do. The penultimate chapter will cover the requirements for the representation of architectural concepts. The ultimate chapter will specify the functionality of a creatrix.

Chapter 2

Critical History

The history of the uses and applications of computers to different types of creative design is short, the first work having been done in 1960. Design considered only as problem-solving (engineering design) by computer is an older subject, but is not as interesting because the process is relatively simple (e.g., determine the depth of this girder with these loads using this formula).

The capabilities of the machines available have strongly influenced the work that could be done. Breakthroughs in hardware give us a way to label periods in the history of design with computers. There are four clear ages. The first period could be called the age of promise when everything seemed possibile, and dreams for the future were expressed. The second age was born with the advent of affordable display devices, and could be called the age of vision. The third age was signalled by the development of virtual memory computers, and could be called the age of memory. We are in the fourth age that was born with single user systems networked for distributed power, when software is more of a limitation of effective use than hardware. Only those of the future will be able to name this age.

In this chapter I will look at events in the history of computer applications to various types of creative design in the first three ages. Historically important computer software systems, influential algorithms, and hardware that affected design software will be covered in terms of goals, computer environment, representation of information, and the commands used to operate on the model.

2.1 The Age of Promise

2.1.1 The Computer-Aided Design Project

The mother of design software systems was work done at M.I.T. in the late 1950's on the Automatically Programmed Tool (or APT) system that let a human operator efficiently convert working drawings into a punched paper "director" tape that would guide the first automatically controlled milling machine. Members of the Computer Applications Group and of the Design Division of the Mechanical Engineering Department met in 1960 to discuss the possibility of using the computer "in a much more direct and powerful way in the chain of events that begins with the original concept as envisioned by the design engineer and culminates in the finished device." [Coons 1963] They envisioned a combination of man and machine using the "creative and imaginative powers of the man and the analytical and computational powers of the machine." They called this concept "computer-aided design." They believed "that the combined intellectual potential of man and machine is greater than the sum of its parts." Two primary objectives were identified [Ross 1960]. One was to develop a data structure capable of expressing relationships between general objects. The second was to describe a language for naturally expressing statements about objects that hierarchically define problems to be solved (in the sense that all substantive problems are hierarchically structured sub-problems). A means of representing all the data about a problem came to be known as a plex structure. It could represent both structure and behavior.

> A plex is considered to be composed of elements of various types, each type of element having a number of components appropriate to the object or relationship which the element represents. [Ross & Feldmann 1964]

These concepts were implemented in a compiled language called AED-0, which was based on ALGOL-60. The Computer-Aided Design System test program was an interpreted language written in AED-0. There was a serious problem of not working at the appropriate level of abstraction. The goals were excellent, the right questions were asked, but they could not work on the essential design issues because too many necessary tools did not yet exist.

2.1.2 SKETCHPAD

Ivan Sutherland did his Ph.D. work at M.I.T. on a now famous program called SKETCHPAD. It is considered to be the first interactive computer graphics program. Its purpose was to demonstrate a method of interacting with a computer "through the medium of line drawings." [Sutherland 1963] The system included a constraint capability based on the structural aspects of the plex concept. The program was written for the TX-2 computer in macro assembler. The user interacted with the program by means of a lightpen to indicate locations, function buttons to select operations and toggle switches to turn states on and off. Knobs were used to scale and position the image displayed on the scope. A keyboard was used only for typing legends. The data, represented in two dimensions, included points, lines and arcs. These could be collected as a "master definition," copies of which could be located anywhere as "instances." Constraints could be placed on the data, such as: lines parallel; lines equal length; line constant length; point fixed; line vertical. Operations on data were performed by pushing appropriate function buttons. There were operations to: draw line, draw circle, move point, join, copy, and more.

A major reason for the successful application of SKETCHPAD to a variety of problems was its ability to work abstractly with complex systems. The user could model geometry and relations and constraints in a general way. There are two important lessons to be learned. Sutherland recognized one himself. "The power obtained from the small set of generalized functions in SKETCHPAD is one of the most important results of the research." [Sutherland 1963] Steven Coons [1963] identified two philosophies of approach to the design of the Computer-Aided Design System: either one provides commands for all the anticipated functions and operations the user might want to perform, or else one provides commands with "the utmost generality." Coons argued that the latter is the better, and Sutherland demonstrated it in his program, but almost all graphics systems today have been built by people who have chosen the wrong path. The second lesson is to understand to power of representing relationships between objects (here expressed through constraints). SKETCHPAD is impressive for its well-thought-out graphics functionality, but it is the constraint propagation through relationships that makes the program exciting.

2.1.3 SKETCHPAD III

As part of his Masters degree at MIT, Tim Johnson used many routines from SKETCHPAD and built a three-dimensional version that shared many features with the 2-D version. It was the first three dimensional interactive graphics program. A method of representing geometric transformations with a 4 X 4 matrix developed by Larry Roberts was used for the first time. The constraint capabilities of SKETCHPAD were not incorporated. The program was built on the same machine and used the same interaction devices. The data structures were essentially the same, but with an extra cartesian coordinate for 3-D. Most of the programming effort went into problems of interacting with a three-dimensional model through a two dimensional image. Four simultaneous views were used, the three standard orthographic projections from technical drawing, and a fourth perspective view. The user would pick a plane to work on in one view, and then add lines with 3-D coordinates by working in another view.

2.1.4 Hidden Line Algorithm

The first algorithm for removing the hidden lines from a 3-D polygonal model was developed by Larry Roberts in his M.I.T. Ph.D. thesis titled "Machine Perception of Three Dimensional Solids." The algorithm was basically a brute force method that is still used (although with extensions that improve speed; for example, using bounding boxes for a quick check of polygon intersection).

2.1.5 HIDECS

Christopher Alexander developed the HIDECS program in 1963. It systematically structured problems that were defined as interactions between user defined elements of the problem. The theory was presented in "Notes on the Synthesis of Form." Because of the static nature of the output, it is usually considered incompatible with creative design, but is quite helpful for a designer that needs to get an initial understanding of a highly complex problem.

2.1.6 Line Rasterizing Algorithm

Bresenham developed a method of representing a diagonal line on devices with limited resolution (incremental pen plotters and raster displays) in 1965. This is a difficult problem because of effects of stair stepping on line width and intensity. Much early computer graphics work dealt with fundamental problems; more and more of these problems are being solved in the hardware today. No doubt this trend will continue.

2.1.7 ICES

ICES is an acronym for Integrated Civil Engineering System. It is a generic program command interpreter (for a Problem Oriented Language, or POL) that manages calls to subroutines that operate on the problem data base. It was developed around 1967 at MIT. STRUDL (STRUcture Design Language) was built under ICES.

2.1.8 Curved Surfaces

Steven Coons developed a method of representing irregular curved surfaces in three-dimensional space in 1967; it is still used.

2.1.9 BUILD

Lavette Teague and Allen Hershdorfer built BUILD in 1967. It was "an integrated system for building design," using constructed using ICES.

2.1.10 URBAN5

Nick Negroponte and Leon Groisser worked on an experimental graphics program to "study the desirability and feasibility of conversing with a machine about an environmental design project... using the computer as an objective mirror of the user's own design criteria and form decisions." As the designer built the spatial model he could specify constraints (e.g., this should not be in shadow) that would be checked by the constraint checker as he worked. A "monitor" would keep statistics on the designer's use of the system, counting rates of the issuing of commands, time in each mode, etc. This information was used to tailor messages to the designer. The program ran off an IBM 360/67 through an IBM 2250 graphics terminal with 8K bytes of display list. The scope was not used dynamically, but like a storage tube. There were 32 function keys, a light pen, and a keyboard for interaction. The data consisted of physical elements (solids, voids, roofs, people, trees, and vehicles) and a site plan. Spatial data (solids and voids) was abstracted to ten foot cubes, each of which had four symbols (circulation, uses, social, and activities) and ten characteristics (aspects of sunlight, outdoor access, visual privacy, acoustic privacy, visability, direct access, climate control, natural light, flexibility, and structural feasibility).

The operations a designer could use were clasiffied by modes. TOPO mode was for manipulating a rectilinear grid with the light pen to approximate the desired site as a warped surface. DRAW mode allowed the designer to control the view of the model and to manipulate the physical elements. The SURFACE mode is used to define cube surfaces as being either solid, partition, transparent, or absent. A surface can also have the attribute of access or not. STORE mode was for file management. There was also the possibility to use the INDEPENDENT modes which consisted of a verb (initialize, qualify, assign, calculate, and display) and a noun (circulation, uses, social, activities, site, elements).

The authors were satisfied with the ten foot abstraction; it wasn't a real problem for the research. They made the mistake of trying to provide all possible functions, instead of generality; when they realized their mistake, they abandoned the project. Some users complained about the "cute" messages (a cube placed in the air brought the response "That is not technologically feasible at this time") that were the result of the authors' attempts to get the system to respond in a human way.

2.2 The Age of Vision

2.2.1 Tektronix Storage Tube Graphic Displays

Before storage tube technology, graphic displays were very expensive (typically $80,000) In 1968 Tektronix brought out the 4010 direct view storage tube graphic display, bringing the price of graphic displays down to $4,000. The displays were not refreshed (the image was not regenerated 30 times a second), but stored the image directly. It was erased all at once like an Etch-A-Sketch, which limited its degree of interaction. The resolution (total number of identifiable points on the screen as opposed to points per inch) was fairly high, but the points were only one color. People in computer graphics consider the pioneers to be those who were working in the field before this device.

2.2.2 BOP

In 1968, G. Neal Harper developed a computer system called BOP (for Building Optimization Program) that would specify an appropriate building configuration based on the client's high-rise office building program. The program was considered as a list of constraints, and BOP would generate schemes and evaluate possible solutions. The solutions were defined in terms of geometry, and were first tested for site limitations, client specifications, architectural constraints, and code requirements. If a scheme passed, it was then evaluated in economic terms in four areas: window wall; elevatoring; HVAC; and structural. A designer could get detailed reports for any solution. There were three types or reports: an architectural summary with dimensions, floor areas and elevatoring; an engineering summary with structural and HVAC information; and a cost and financial summary with costs by trade and type, and financial information.

The program was run on an IBM 1130 8K computer with a disk drive, a card reader for input, and line printer for output. There were about thirty component items with one to seven variables each. The data representation was comprehensive. The variables dealing with geometry included: length, width and height of floor; dimensions of the core; lease span; and building module. There were also variables for window wall specs, elevatoring, mechanical systems, structural systems, building code information, and economic data for both costs of materials and investment data. There were "reasonable values" for all of these, so the designer could use BOP without knowing everything about the program. The defaults would be used for the unspecified information. The method of interaction was simple. A user would type commands on a card punch and have them read. There were two basic types of commands: those specifying the constraints implicit in the program; and those initiating evaluation and printed reports.

Harper considered the practical aspects of building design to be an optimization problem. BOP was unique in that it treated the total problem of practical design, rather than just an isolated part of the problem in more detail. BOP was highly used, but not for what it was designed for. The designers preferred to do the scheme generation themselves, but liked to use BOP to get an elevator specification and for an economic analysis. It became a trick to understand BOP well enough to be able to specify the variables that would cause BOP to generate the scheme wanted in order to get the analysis. With modern computers, BOP could be reproduced with a spreadsheet.

2.2.3 First Computer Graphics Applications Companies Founded

In 1968 Dave Evans and Ivan Sutherland formed a company to market computer graphics hardware. In 1969 a number of companies were started to market turnkey design and drafting systems (primarily for engineers). These included three of the biggest: Computervision; Intergraph; and Applicon.

2.2.4 DISCOURSE

DISCOURSE was an interpreted computer language that accommodated the design of large scale urban environments written in 1969. It could be considered as a programmable 2-D raster based thematic mapping system with constraints. It was not a system for the communication of ideas to others, but for a *discourse* between the designer and his ideas. Aaron Fleisher originally wanted to simulate design with the computer. This proved to be too much, and William Porter developed it as a design partner. DISCOURSE considers three types of functions that represent the activities of a designer: functions of further description, functions of transformation, and functions of testing.

DISCOURSE was written in AED and ran interactively under the CTSS time sharing operating system from a teletype, with the verbal and graphic output showing as printer graphics. Basic data comprised a geographic grid of cells or locations (each representing an area) and a list of ATTRIBUTES (each representing a list of locations with shared characteristics). ATTXLOCs were ATTributes which eXisted at a LOCation. ATTRIBUTES were defined with CHARVARS (CHARacteristics that VARy at a location) and CHARCONS (CHARacteristics that are CONstant at any location). There were commands for adding and modifying data (either specifically or conditionally), and for defining attribute lists.

Manipulations dealing with grids fell into four categories: existance at, proximity to, adjacency with, and orientation of. Two difficulties were identified in the design of DISCOURSE. One was in understanding how a designer works, and what vocabulary and grammer are needed. The other was in knowing what the division of labour between designer and computer should be. It was noted that "subtlety of expression and ease of use may be opposed." One problem with the system was the amount of time spent writing macros (called RCOM files, for Request COMbination). This indicates that the designers wanted higher level commands. Another problem was the constant need to refer to locations by their coordinate pairs. This was considered unnatural for designers, and inconvenient.

2.2.5 Watkins Algorithm

A hidden surface algorithm that would display a shaded image of a polygonal model was developed by Watkins in 1970. It performed its calculation in image space working one display scan line at a time.

2.2.6 Gouraud shading

A shading algorithm for the smooth shading of polygonal representations was developed in 1970. This allowed curved surfaces to be represented more efficiently as polygons, and rendered and displayed as smooth surfaces. The technique is to calculate average surface normals at the vertices and to interpolate the intensity of the color between edges.

2.2.7 ARK-2

(Architectural Kinetics-Man and Machine) One of the first commercially available architectural graphics packages. The goal was to develop a system of programs that could be used to study feasibility, for programming, in preliminary design, design development, working drawings, specs, supervision and office management. In 1971 the programs ran on a DEC PDP 15/20 with 16K words of main memory. Interaction was through a Computek vector display and a digitizing tablet. The basic elements were lines that could be collected in subpictures that make up drawings, and Standard Graphic Elements (SGE) which were things like construction details and furniture. There were commands to create a new drawing or database, edit an existing one, and to store or retrieve files.

2.2.8 OXSYS

OXSYS was a comprehensive system for use with the Oxford Method of Building. It ran on an Atlas II computer in 1972. The design rules that related the components of the building system were stored as data, rather than built into the computer program. This made updates to the building system easy and the code writing fast.

2.2.9 IMAGE

The IMAGE system, for space layout by constraints, used a problem oriented language to interactively develop floor plan arrangements. Tim Johnson et al. considered spatial arrangement to be a central task of preliminary architectural design. This seemed to them to be a good place to begin in developing an aid to allow architects to explore more variations. The other goal of the research was to explore and clarify the design process. In 1972 the IMAGE programs ran on an IBM 360/70 time sharing computer connected by phone to an IBM 1130 which was used to drive an IBM 2250 dynamic graphic display (used statically). Data was stored in the SKETCHPAD III data structures. Spaces were represented as orthogonal boxes (that could be displayed rotated, or as representative volumes with more detail than boxes) defined by location, size, and orientation. Composite volumes could be assembled for more accurate representation of spaces. Eleven constraining relationships were defined which could be applied to the spaces with differential weighting: these were: non-overlap; reshape; overlap; proximity; alignment; relative position; visual access; circulation; area; proportion; and dimension. IMAGE generated arrangements by modifying the spaces so that the constraints would be satisfied to the degree possible using least mean squares.

Problems with IMAGE were its rigid form description capability, and the requirement that the designer define the problem in terms of the eleven relationships. Users felt they gained insights into the design problem by using IMAGE that they wouldn't have gotten otherwise.

2.2.10 GSP

GSP was developed by Charles Eastman to be a "combination interactive and automated architectural problem solver and drafting system." It automated the generation of 2-D floor plan arrangements. In 1973 GSP could run on either an IBM 360/67 or a PDP-10. It was written in FORTRAN IV and required 50K words of core memory. Interaction was through teletypes, storage tube terminals, and plotters. The basic spatial data element was called a DU, for design unit. There could be eleven of them. They could be built up from rectangles (but were treated as their bounding rectangles), and could be classified as either empty, solid, or circulation. Four relations could defined between the DUs: adjacent; sight; distance; and orientation. There were command modes for shape definition, relation definition, interaction, and automation.

2.2.11 Phong Shading Algorithm

A shading algorithm for realistic smooth shading of polygonal representations was developed in 1973. The technique is to calculate average surface normals at the vertices, interpolating the normals between edges, and calculating the lighting for each pixel.

2.3 The Age of Memory

2.3.1 DEC Virtual Memory Computer

2.3.2 DEC VAX 32 bit virtual memory computer 1978

The Digital Equipment Corporation developed the VAX 11/780 super-mini multi user computer with a 32 bit bus giving it a large address space, and virtual memory removing the limitations of the size of main memory.

2.3.3 Ray-tracing Algorithm

In 1979 Turner Whitted and Kay & Greenberg published papers on techniques they had developed to realistically render images of computer models of physical objects. The remarkable aspect of ray-tracing is its ability to accurately render the interaction of light with a modeled environment, including shadows, reflection, and refraction. The most realistic images produced have been made with this method, but it is very slow, even in an efficient implementation. There are still no commercially available programs.

2.3.4 PLUS-3D

A general purpose architectural 3-D graphics system was developed by Skidmore, Owings & Merrill. It was written in compiled BASIC and ran on PDP 11/70s, and VAX 11/780s. Interaction was through Tektronics 4014 storage tube terminals. The basic data elements were points, lines, polygons, and text strings. These could be collected in groups. It used a hierarchacal menu. There were commands for viewing, data manipulation, and geometric calculations.

2.3.5 Intergraph APDP

Intergraph released several computer programs for architectural work in 1980. Their goal was to market a system that would improve communications between the architect, the engineer, and the space planner. The programs run on VAXs using large workstations consisting of a large Rand tablet and two large monitors. The basic data elements are points, lines, a variety of curves, and collections of these that represent doors, windows and so on. These can be linked with a data base. The main program is APDP (architectural presentation and design package). It includes systems for floor planning, reflected ceilings, site plans, elevations, details, and sections, all in two dimensions. There are also programs for space planning (that uses APDP floor plans), MEP / HVAC, strategic planning (manipulates block diagrams with square footages for adjacency and stacking diagrams), spec generation (assigns paragraph numbers to drawings), and architectural modeling. The last is the only one that works with 3-D data. It has commands for the placement of walls, windows, and doors. It will generate hidden line and hidden surface images.

Chapter 3

Representation

If you want to know what an architectural idea somewhere in your imagination will really look like when it is built, you can build it and see. This can be expensive. If the idea is of an unstable structure, it can be dangerous. If you include detail extraneous to the original idea, the concept can get lost. It can take a long time to build. If you want to have someone else build it, you have the problem of expressing your ideas to the builder. It makes a lot of sense to make a drawing of the idea, or to build a scale model.

3.1 Drawings

The basic tools of conceptual/creative design are pencil, eraser, and paper. They can be used to create diagrams of relationships and processes, to do sketches of vague ideas and plans, to do drawings of real or imaginary objects, and to render views of these objects.

Paper and pencil have five advantages in how they represent ideas. They can be used to express ambiguous ideas. A drawing of a space might be represented as square and rectangular simultaneously. Irrelevant aspects of the world are easily ignored. A diagram of circulation can ignore the building's structure. Variable detail is possible. One part of a drawing can be schematic while another is drawn in detail. Impossible situations are representable. A drawing does not suffer the laws of nature. The last advantage is that they allow ideas to be expressed and changed easily. Lines can be drawn that are not part of the representation, but exist only to ease construction of the image. The images can be modified by erasing or drawing darker, or by tracing.

Drawings have a disadvantage. They represent a static view of some physical or conceptual world. When a designer wants to understand a dynamic aspect of the world (such as what it is like to walk through a space, or what is the progression or the sun's shadows) he must build a three-dimensional model.

Since a drawing represents an imagined or actual object, it is a type of model. Sketches and drawings can be diagrams, plans, or renderings. Diagrams usually model relationships or processes, plans model configurations, and renderings usually model visual characteristics. The word "model" will be used to mean any representation that shares characteristics with that which is being modeled. By definition, a model of an object has lower informational content than the object itself, otherwise it must be the object. The only perfect model is the thing itself.

3.2 Models

By looking at the characteristics models represent, and at what they are used for, three types of models can be identified.

Analogical models are used to make inferences. They are based on the assumption that weo things known to share some characteristics, may well share others.

Theoretical models are used to predict and explain relations and causal connections between facts.

Conceptual models are used to understand and represent ideas about a world. They do not necessarily have a known analogue so inductive inference is not a factor. They cannot be used to predict and explain phenomona because there is no hypothesis. They are more descriptive than predictive. Not surprisingly, conceptual models are especially useful in conceptual design.

Conceptual models are important to formal design for two very important reasons:

A conceptual model provides a means of *organizing* knowledge. This is not an important issue for a mannerist, but it is for a designer working with fundamentals.

Architecture involves a large number of elaborately interrelated elements, both in terms of composition and in terms of construction. Conceptual models can simplify this complexity, making projects more manageable. This happens when relevant details are structured, and others are ignored.

A conceptual model also provides a means of *externalizing* ideas. An explicit expression of ideas is needed for a designer to objectively evaluate them. (This is related to Clive Bell's idea of "aesthetic distance.")

3.3 Abstraction

Plato explained that all earthly objects were imperfect copies of the ideal objects in heaven. We can recognize two buildings as both being houses because they both represent an abstract house in heaven, which is a pattern or ideal. An abstraction is an idealized version, a general form or idea "in each class of many particulars to which we give the same name." [Plato 350 BC]

The process of defining an abstraction is very similar to the process of modeling, and in a sense they are the same. Both involve decisions of which characteristics are important and what descriptive formalism to use. But this discussion only holds if the object being modeled exists only in the designer's mind. Otherwise he is making an imitation of a designed bed, an imitation in the second generation from nature.

The designer defines a new ideal, an abstraction in heaven. To represent design we need to add two levels of reality to Plato's list.

```
IDEAL - THE ABSTRACT IN HEAVEN
+ designer's idea
+ designer's representation of the idea
WHAT THE CRAFTSMAN MAKES
WHAT THE ARTIST PAINTS
```

Some architects work at the artist's level, reinterpreting the existing. Other architects work just below the ABSTRACT level, and others inbetween.

There are different levels of abstraction that differ in how true they are to a particular instance (the reality of that which is represented). A model of an idea for a walnut table might be made full size in pine. It will not accurately represent weight, hardness, or color, but will display proportions and dimensions faithfully. A small block of walnut might model the table at some scale, but would not share many specific characteristics. It would embody less specific information than the pine model or the "real" table.

For different purposes, yet always representing the same idea, a kitchen might be modeled as a circle drawn with pencil, a cut-out square of paper, a block of wood, and so on. If the nature of the kitchen is known, then these abstractions are different aggragations of detail, otherwise they are more or less particular or specific definitions.

By being less specific to a particular instance, abstraction can be a barrier on detail. It might 'hide' details that are, at some level of work, not relevant to an understanding of the object. The Platonic form *table* doesn't define the connection of the leg to the top. In this sense it is an aggregation of detail.

All elements of a building interrelate. Chairs and tables relate to walls and doors. N. John Habraken has discussed these relationships in terms of dependencies between levels. Tables are dependent on walls because if the walls change, a change is forced on the table. These dependencies are representable as a hierarchy.

Elements all at a given level also interact, but not through dependencies. Their relationships are not so simple or unique. Different rooms can relate in how people get from one to another, acoustically, thermally, or in other ways. A graph or network can represent these interactions.

3.4 Meta-Models

Drawings communicate. Figural or formal drawings represent ideas. The idea may or may not be represented in great detail, but often a large portion of the information is in how the idea is drawn. The information in a drawing is not just in what is represented, but also in how it is represented. A hard line drawing suggests definite ideas; fuzzy lines imply uncertain ideas. A drawing not only describes some idea or world, but also communicates information about the description of the idea, information about the knowledge of the model.

Charcoal and 3B leads are most often used to represent uncertain ideas and their more ephemeral aspects. Technical pens and 3H leads are most often used to represent well thought through ideas.

When a drawing (or model) is the direct result of our own manual efforts, we have a good idea of not only what world information we are trying to communicate, but also what information the drawing communicates about what it is we know about the drawing (as representing information about the world). We know what ideas we have about the world, and we know what we know, what the nature of those ideas is. These things are all expressed in a good drawing, but often fail to come out correctly in computer generated images. People seem to trust computer output, even when there is no reason to. Computers are so precise, people think they must be accurate, also.

There are several dimensions to measure when describing a drawing, as opposed to describing an idea or world:

The first measure of a representation is how abstractly the idea is expressed. Does it represent an idea apart from a particular object, or does it specify details?

The second measure is how ambiguously the idea is represented. Does it have several possible meanings, or is it specific in representing one? Ambiguity is an important aspect of conceptual sketches, allowing an architect to express several ideas without committing himself.

The third measure is how accurately the idea is represented. Is it true to the designer's intentions? This can be hard to read in a drawing, but usually a sketchy drawing is less accurate. Some aspect of a drawing might be completely accurate, but not absolute; e.g., a drawing of a pullman kitchen is accurate if it is going to be a pullman kitchen, but it is not absolute because there are a couple of places it could go. Working drawings are often inaccurate, but only at a level of detail presented to support an accurate definition at a higher level. Good draftsmen know not to spend time drawing irrelevant details accurately.

The fourth measure of a representation is how complete or absolute it is. Are they final, complete, and certain, or are they tentative and preliminary? In a handmade drawing, the measure of how close to final the ideas represented are is the vagueness of the expression.

Chapter 4

The Design Contingence

Every problem contains and suggests its own solution. Don't waste time looking anywhere else for it. In this mental attitude, in this mood of understanding, lies the technical beginning of the art of expression.
Sullivan in Kindergarden Chats (section XLIX)

Almost every article and book written on the subject of the use of computers in design has a section or chapter entitled "The Design Process" where the author explains what design is. With some the definition is very general, for example "turning ideas into reality."[1] With others it is more specific, for example, "devising a course of action aimed at changing an existing situation into a preferred one."[2] Unfortunately, it is necessary to have a good idea of what "design" is in order to design a computer system that can be used to design with.

People that try to define "design" as a process always seem to push the creativity into another spot. For example, there are those who say design is constraint satisfaction; then the creative aspects of design move into the specification of the constraints. Because creative design involves inspiration and is therefore nondeterministic, here it is considered a contingence[3] and not a predetermined process. Design as a process is an important concept for many aspects of design, but fails to catch an important element of conceptual design, namely serendipity.

[1]Going to visit a friend after having the idea would be design.

[2]Setting up a diet to loose weight would be design.

[3]"The coming to pass of anything without predetermination,... happening by chance." [Oxford English Dictionary]

Different schools of thought have different theories of the nature of creative thought and design. Within the design professions there are major disagreements as to what design is. Some designers believe that design is problem solving, others that design is the application of creativity. I will differentiate between what can be considered different aspects of design, so that seemingly opposing views can be shown to be appropriate in different design situations.

4.1 Aspects of Design

When design works with information that expresses social values through the emphasis on exaggeration of aspects of the object or pattern, we can call it "stylistic" design. For example, women's cloths have recently put an emphasis on the ankles and shape of the leg with short, tappered pants and short socks. This expresses the greater emphasis people are putting on the importance of women looking feminine. Design of this type involves application of external information or rules; that is what makes it style, rather than idiosyncracy. When the style is popular, design becomes the application of rules to that which is being designed. When the style is personal, such as Wright's prairie houses, it might have been developed or adapted (from the Japanese, possibly), and design becomes the expression of ideals.

When design works with information that expresses knowledge of the physical world through technology and its laws and formulas, we can call it technological design or engineering design. For example, in the design of a reinforced concrete column, there are established acceptable ratios of cross-sectional area of vertical reinforcement to the gross column area, and there are formulas to calculate the maximum permissible axial load on columns. Design of this type involves application of coherent knowledge to identifiable problems.

When design works with information that expresses proportion, scale, color, texture, rhythm, and harmony, we can call it aesthetic design. For example, in the design of rugs, tile patterns, facades, columns, there are aspects that are detailed for functional considerations, but there are others that are arbitrary in many respects. Design of this type involves application of taste to identifiable problems.

These types are not meant to classify, but to describe. Design is not well enough understood to define an accurate taxonomy. The types of design are more like the colors in the spectrum. Red is a description of a color we see, but the label can be applied to many colors, and there are many unnamed colors in between the named ones. The design types are similar; there are other types that can be named in between the named ones. In between the stylistic and the technological, there is a type of design that might be called functional. It gives form to objects that are used for practical purposes. That between technological and aesthetic could be called morphological design. This type manifests considerations of internal or essential form and structure. Between the aesthetic and the stylistic is poetic design, which gives expression to the designer's or artist's emotions.

In some situations design can be considered as primarily problem solving; at other times design is best considered as problem finding; at still other times design starts when the problems are solved.

4.2 Design as Problem Solving

Design as problem solving is a common and useful simplification of an activity that is difficult to understand. Problem solving can be defined as purposive behavior aimed at achieving a goal which is not trivially attainable. The goal can be a route or method, a situation or state, or an object, real or abstract. It would be nice to report that the problems of problem solving have been solved, but they haven't. The techniques used for problem solving are of two types. When the knowledge about some domain is systematically complete, algorithmic stratagies can be used; problem solving is relatively straightforward. When the knowledge is vague and incomplete, artificial intelligence techniques can be used; problem solving is a real problem. Problem solving will be considered in terms of what can be automated.

4.2.1 Well-Defined Problems

When the knowledge about the domain that the problem is framed in has a high degree of coherence, solving problems is simply a matter of identifying the problem and then applying a known solution procedure to the relevant facts. The problem in solving well-defined problems is defining the problem to be solved.

Well-defined problems can be solved automatically through the use of algorithmic solution procedures. There are several basic techniques for problem solving with well-defined problems. The simplest is commonly called "plug and chug." The appropriate values are substituted in an equation, and it is solved.

Complex problems must be separated through decomposition, and the parts solved separately by the divide and conqour method. Alexander tried to formalize design as problem solving through decomposition [Alexander 64]. He later abandoned the technique.

4.2.2 Ill-Defined Problems

Problems can be ill-defined for two reasons: the solution procedure may not be known, or all the information required for solution may not be available. There are two paradigms from artificial intelligence technology that can be used in these situations: search and inference. Each requires a systematic representation of the problem to be solved. If a complete enough representation of the problem system cannot be created, simple problem solving is not possible.[4] This just means that a problem cannot be solved if the problem cannot be identified well. If the solution procedure is not known, then search can be used, if there is a way to recognize a solution. This strategy is effective when the goal can be characterized as a sequence of choices. This mode of designing is characterized by final selection of alternatives through testing.

The need for a good search strategy becomes obvious if you consider an example. If the problem is to pick a different one of five facades for each of five buildings in a Hollywood set, there are 5! (120) possibilities. If you try to check each one, and can check one a second, it will take you two minutes. That may not be a problem. But, if there are ten facades and ten buildings, it will take 42 days. For twelve, it will take fifteen years nonstop. This is called combinatorial explosion.

This mode of problem solving can be described as a process of searching through alternative states of the representation in order to find a state that satisfies predefined criteria. The collection of all the possible states of the representation is called the state space. A state space is usually diagrammed as an inverted tree. The initial or current state is the root node, and the succesive state nodes are connected to a state node by lines called branches.

Search strategies can be divided into three groups [Wins 84]. Some searches

[4]How many angels can dance on the head of a pin? It can't be calculated without a complete representation of angels.

find some path or sequence of steps, but not always the best. Other searches find the optimal path. The third group of searches are game searches used when there is an adversary. The best search to use depends on the nature of the state space, and on the ability to make judgements in progress.

Inference can be used if there is a way to deduce facts from the known facts. This strategy is effective when the knowledge can be expressed as axioms. This mode of designing is characterized by trying to make a statement about a desired situation true by using operators to change things. Three types of inference are identified in [Char85]: deduction, induction, and abduction.

Logically correct inferences can be made using deduction (where particulars are derived from general principles), as long as the axioms are true. Deduction is usually represented in a system called predicate calculus, which calculates the truth of propositions.

Rule-based expert systems are often deduction oriented. They are called forward chaining if they work from known facts to new, deduced facts. A backward chaining system hypothesizes a conclusion and works backward to supporting facts.

Plausible inferences that are not necessarily logically correct can be made using induction, where general principles are derived from particulars.

Another type of plausible inference that is not necessarly logically correct is abduction. Expert systems have been built based on abduction using Bayes's theorem. It states that knowing the percentage of situations with some attribute that happen with the situation, and knowing the likelihood of the attribute and the situation, then you can figure out the conditional probability of the situation given the attribute. If there are very many attributes, then the number of percentages that Bayes needs to know become astronomical.

4.3 Design as Problem Seeking

Problem solving is only possible when there is an identifiable problem and there are identifiable criteria. If there is neither a problem that might have a solution nor a way to test a proposed solution, then problem solving techniques cannot be used. This situation occurs when a challenge is made, or a dilemma is presented. The designer must find the problem to be solved before he can solve it. "Design an office building for me that expresses my company's commitment to quality." Ahh, this is the stuff of imagination and creativity. This is a problem, but one of a different type. This problem must first be found or identified.

Somehow, the problem is to express ideals, values, emotions, and impressions. This is the kind of problem that architects usually talk about when talking about design problems. It is fundamentally different from a problem that is solved by satisfying constraints. Here the problem is a goal that is like finding a shore line, the closer you look, the harder it is to locate precisely.

The course of designing includes learning about the project and discovering new ideas that seem important and ways to express them. Opportunities present themselves. The originally fabricated constraints can then be modified or changed. The learning aspect of design is very important; it helps the architect to produce a more highly resolved and better building. The act of drawing is a very good way to learn about a project. It forces the architect to give attention to aspects he might not have thought of otherwise. It provides time for the designer to consider his schemes. It informs the architect, and makes its own suggestions of what to do.

In practice there are always some constraints on a project: possibly the site, technology, the local ordinances, and the use. But there are major aspects of a design that are completely unconstrained in a problem sense. "I want a house on this site that affirms my belief in the importance of family." Some architects have said that design starts when the problems have been solved. But designing changes the quality of the problem solutions.

4.4 Sources of Concepts

Architects given carte blanche, as is often the case with contests, have to first define constraints that are basically arbitrary. Some conceptual framework is needed to start. The program might do this if it is fairly thorough. Otherwise a source of concepts is needed. If the designer is a neo-mannerist, there are precedents. If the designer believes in a geometric system, there are prescribed ratios. The designer might try to find the system that the project itself inspires, what the building itself "wants to be." The actual source of new concepts is difficult to understand. There are probably many. Some have been identified as possible. There are excellent discussions in [Wertheimer 49] and [Schon 63].

There are the sources that are basically mysterious: the Muses whispering, strokes of genius, a flash of lightning, visions, and intuition.

Rationality and logic are reasonably understandable. From known facts or ideas new concepts can be inferred.

Another source is the recombination of old ideas. The new ideas are the ones that get through a filter that only passes good ideas that result from the combination of mental elements.[5] Young designers often have the problem of filtering out too much. Older designers are often inspired by the "random" ideas

[5]This is known as associantism by Wertheimer.

that come out during project exploration; once again the idea of learning from the project arises.

Another source is the structuring of *gestalten.* This emphasizes the minds need to structure concepts as wholes so as to be easily comprehendible. It is in a sense an extension of gestalt theories of visual perception. It is basically a mysterious source which Wertheimer calls "seeing the light."

An important source is in the displacement of concepts [Schon 63]. This is the use of concepts as metaphors for new concepts. The new concepts evolve from old ideas. Not just new concepts, but architectural schemes, too, evolve. A scheme acts as its own old concept to inspire new concepts. Design becomes a reflective dialogue with the materials, an idea discussed in [Scho83]. This is again an example of the importance of learning in design.

Chapter 5

Instruments of Conceptual Design

A creatrix is a computer system for the representation and manipulation of the ideas and concepts of design. It surpasses paper and pencil techniques by maintaining a unified description of both abstract and concrete aspects of a project that can exist with varying degrees of abstraction, ambiguity, accuracy, and completeness.

If the creatrix were a collection of integrated computer programs for working with program analysis, site analysis, building design, mechanical design, and structural design, then it could be defined in terms of these activities. But the creatrix is a general purpose concept representation system. It could be defined in terms of how it deals with the phases of architectural design (in the AIA's definition), pre-design, schematic design and design development. But these phases don't reflect the creativity involved. Because it is being specified to assist in the nebulous task of conceptual design, it is impossible to say what it is that it does in terms of specific tasks, but its capabilities can be described by analogy.

The best description of a creatrix is as an instrument. Just as musical instruments give a voice to the musician's ideas, the creatrix lets the architect express architectural ideas. Just as mathematical instruments allow mathematicians to work with complicated ideas, the creatrix helps the architect to manage complex architectural ideas. Just as scientific instruments increase the scientist's ability to quantify and qualify, the creatrix increases the architect's abilities to evaluate architectural ideas.

5.1 Vision Imaging

The creatrix is like a musical instrument because it provides a means for the designer to express his ideas and "see" them in the "real" world.

A basic aspect of conceptual design is the graphic representation or imaging of ideas and visions. This is important for several reasons. Vision imaging is important and often necessary to effectively explain the designer's ideas to both himself and others. It is also needed for the creative dialogue with the Muses.

There is a danger in thinking of representations as models. It might indicate that simulations are possible, but there might be no link with reality past the image that is created. Drawings can be no more than beautiful images, and these can be used in design [Stoker 86].

A very difficult part of a creative person's work is finding the ability to make the objective judgements of a creation that are necessary to achieve quality. We become emotionally attached to a piece of work and want to see it survive, even when it is preliminary and we know that it needs major revisions or development. We see what we like about the scheme, and don't see the rest.[6] A good designer must know how to burn idols. It helps to be able to stand outside the temple and see as an outsider sees. A creatrix can help by showing an idea from different perspectives, with different lighting, and so on.

Seeing an image of an idea rendered with shadows is an excellent way to see a scheme as an outsider. Specifing a day and time, and then seeing a scheme as it would appear at that time gives the concepts a reality that helps make it easier to burn idols. Sun lighting allows a scheme to be seen in a new light.

[6]Conceptual design sketching is like writing love letters, one's in love and is blind to faults, and just wants to dwell on the romantic aspects.

It is usually hard to hear what the Muses whisper. They have soft voices and don't enunciate clearly all of the time. They don't usually repeat themselves. They often speak in a language we don't understand very well. We can often coax them to say more by sketching what we hear. The designer has to be able to express these often ephemeral ideas to a creatrix effortlessly because the Muses don't wait, and leave when we don't give our full attention.

The creatrix will have to be able to understand sketching as representing three dimensional form, without the designer giving commands to make up for the two dimensional input that are unnatural to thoughts about the design. Sketch recognition is difficult, but new work shows that there is a technique that is viable [Jurgensen 1986].

The geometry of an object can be defined in relative to a few key dimensions, and variations can be compared by changing the values of the keys. Parametric variation can also help the designer burn idols because a scheme is seen in new ways.

The imaging of visions is important not only for questions of form in space, but also for any questions about concepts that can be represented graphically, which includes verbal notes. The ideas might be of an arrangement of rooms in terms of circulation, or a way to relate the building to its context.

It has been assumed by many that realistic images are needed for computers to be useful in design.

> ...available tools are simply not flexible enough to use for preliminary design and testing of alternative strategies. Nor is the output realistic enough to permit esthetic or design evaluations. [Greenberg 1984]

But the issue is not realism, it is density of relevant information. With some ideas, very little realism is needed, possibly only the size and position of a shape. With others, more is needed, for example lighting and shadow studies.

A designer should be able to improvise on a creatrix, composing space on the spur of the moment. Then we will have improvisational architecture. Architects will get together to jam. The test of the creatrix will be: can a designer[7] sit down and play, or will they laugh? Music is liquid architecture.

5.2 Concept Management

A creatrix is like mathematical instruments[8] because it helps a designer to manage complexity.

A difficult aspect of conceptual design in architecture is keeping track of the needed rooms, their areas, relationships, and characteristics. It becomes more difficult when their representations are changing. There are two aspects to the management of concepts. One deals with the abstraction of details, the other with the structure of the representation, which includes relationships and constraints.

It is important to be able to represent ideas with any amount of detail that is appropriate. The amount changes as the design progresses. Initially, the objects might be defined only as existing, and their relationships are studied. Some objects might start out completely known. Objects are often defined in terms of sub-objects that may themselves have sub-objects. Sometimes lines are drawn to communicate feelings without actually representing anything specific. These don't fit into the scheme's structure, but are important for stimulating ideas.

In order to represent relationships between objects, their representations need to be structured as a network. Because there are many types of relationships (e.g.: adjacency, visual, acoustic), several networks need to exist concurrently.

[7]One experienced with the use of the creatrix.

[8]An example is the calculus, which makes complicated processes, rather then states, managable computationally

Related to relationships, constraints provide a means to encode information without committing to a specific form. They allow conditions and relations to be defined in abstract terms. SKETCHPAD was a constraint based system [Sutherland 63], as was IMAGE [Johnson 70]. New work has been done applying constraints to a 2-D graphics editor [Nelson 85], and to design [Gross 86]. I see constraints as providing a means of representing knowledge about characteristics of a desired solution, rather than a means of finding the solution.

There is a very powerful design technique[9] using dual models that solves the idol burning problem. A scheme is initially proposed. Then the designer attempts to correct its flaws and weaknesses in a second model. The two representations are then studied and the poorer of the two is replaced in a new iteration. It is always possible to experiment with the weaker, until it becomes the stronger. Radical changes can be made with complete safety, and if one looks promising without beeing better than the current stronger, a new pair can be developed, later to be compared with the other strongest. It is a contingence similar to the hill climbing process in problem solving, applied to problem seeking.

5.3 Scheme Measurement

A creatrix is like a scientific instrument because it helps a designer to quantify and qualify a scheme.

An necessary aspect of conceptual design is checking that a scheme works well. Measurements and calculations can be made on technical, relational, and geometric aspects of a building. A creatrix will deal explicitly with these three.

[9]I first saw it in a course developed by Walter Peterhans, an associate of Mies.

There are several technical considerations that can be measured using rules of thumb and approximations. Accurate analysis is beyond the scope of a creatrix, and really isn't usually appropriate during conceptual design. Daylighting and natural light levels are important considerations that are often left to the detail design. Few architects consider acoustics specifically, but often should. Energy and insolation calculations should be made early because fundamental form differences affect the results. Architects usually have a good sense for what is structurally sound, but more information early on can lead to better solutions. These measurements are important during conceptual design because they can have a profound influence on the final design proposal. It is often difficult to make the major changes that the calculations can indicate are needed when they are made on an almost final scheme.

When a description is represented as a network, a number of measurements can be made. These can be used to find the shortest path, the Hamiltonian path, minimal-cost network, and other things that architects don't realize they calculate. Clustering algorithms and decomposition can be used for project analysis.

Measurement of the geometric aspects of a scheme is essential. The basic measurements are: length and distance, area and surface, and volume. Angle measurements are also needed.

Chapter 6

Representation of Concepts

It has been said that the goal in designing a representation is to find the essential structure that is present across a class of problems and to provide those structures as primatives. This is fine when working with programs that will solve specific problems in a clearly defined domain, but fails in an unclearly defined domain for two reasons: First, a needed structure hasn't been provided because not all needed structures were anticipated. Second, the structures are too basic for efficient use. The real goal in the design of a knowledge representation scheme is to match the system description to the designer's conceptualization of the domain, and to provide the means for the designer to modify the system description as the conceptualization changes.

There are many levels of representation possible. Examples of three of them are: a representation for some type of room, a representation for the elements of a room, a representation of points and lines that represent an image of a room.

There are also many directions of representation possible depending on what aspect of reality we want to model. One method might be to represent a room as an air volume and thermal mass; another might be to represent a room as a node on a circulation network.

Many architectural computer systems have been built without incorporating any real architectural knowledge. They are called drafting systems. Current graphics systems are just now passing the point of only representing points, lines, and polygons; and are starting to give collections of these the attributes of doors and windows. This added information is first order architectural knowledge. It represents objects, their parts and characteristics. There are two classes of techniques for object encoding: the declarative and the procedural. This degree of knowledge is not good for much more than quantity take offs, although preprocessors are often written to massage the description for a specific analysis. Second order architectural knowledge represents the relationships between objects. This means a computer can know more about the objects, which means higher level commands can be used by an architect to manipulate the representation.

This chapter does not describe the details of a specific representation scheme, but the rationale. A specific proposal for a scheme that incorporates most of the ideas here is in [Jurgensen 86b].

6.1 Sources of Information

There are many sources of information that together form the body of knowledge for a project. These sources need to be recognized if a computer representation is to be defined:

Building codes and zoning ordinances are known through laws. This knowledge includes information on allowable room sizes, travel distances for fire escapes, possible construction materials, and so on. These things are often somewhat ambiguous, allowing goverening bodies flexibility.

The limitations of materials are determined by technology. This knowledge includes information on the modulus of elasticity for a steel, properties of a caulking material, and so on.

Terrain, climate, and context are determined by the site. This knowledge includes information on soil conditions, water table, solar exposure, prevailing winds, and so on.

Needs and wishes are described by the client. This knowledge includes room descriptions, their interrelationships, cost limitations, desired materials, and so on.

All of these are constraints on the design of the project. They are constraints on what is built, and on how, but they are rarely strictly inflexible. The building code can be gotten around with a variance, new materials can be developed, sites can be graded, and clients can change their minds (both through whim and through discussion). These four sources of information are all external to the architect. There are two sources which, in a sense, come from within the architect, and define the project itself.

Personal style and strataegy are defined by the architect. Examples are the *parti* or organizing principle, the aesthetically preferred materials, and so on.

Good solutions are discovered. Examples are: the best proportions for windows and their placement, a beautiful entry approach, and a good way to seperate the public and service areas from the private.

Competent architects have a good working knowledge of legal requirements and material limitations for conceptual design. An expert system could be built into a creatrix that would allow it to monitor the design for code infractions, but there would be many problems. The system could hardly be expected to make judgements on a scheme that is incompletely or ambiguously expressed. This sort of computer program would be fine if architects only dealt with fairly absolute designs, but they don't. The creatrix is to help manage information, not supply and synthesize it. Since the internally defined and discovered knowledge incorporates the relevant aspects of the first three external sources, the creatrix primarily needs to be able to represent knowledge from the last three sources. Aspects of the site description are a notable exception.

6.2 First Order Architectural Knowledge

There are two basic methods of representing knowledge. Information can be encoded as declarative knowledge, or as procedural (also called imperative) knowledge. Declarative knowledge says the circumference of some circle is 25 units, while procedural knowledge says the circumference is the diameter of the circle times *pi*. A piece of declarative knowledge is some information, while a piece of procedural knowledge is a means of generating some information. The declarative is a dinner while the procedural is a recipe for a dinner.

6.2.1 Declarative Knowledge

Declarative knowledge is typically stored in tables or a data-base. There are several advantages to this method of representing knowledge. It is much easier to encode unordered information, like a site description, and it is also much easier to encode unique data, like the Barcelona Pavilion. Since the knowledge is separate from the program that operates on it, it is much easier to modify the information and it is much easier to extend the information at a later time.

There are three basic methods of storing declarative information: as a network; as a hierarchy; and as a relation, an array or table. (Hybrid systems are combinations.) These translate to three common methods of representing graphic information.

Constructive solid geometry (commonly called CSG) represents objects as a network of primitive forms connected by arcs that represent Euler operations (union, intersection, and difference) on the primative forms. When the network is analyzed, the result is some more or less complicated shape. This system is used primarily in engineering analysis of mechanical parts because mass properties are effectively encoded.

Another method is based on hierarchy. It is called octree representation. The top node represents the volume the object to be represented is enclosed in. There are eight nodes beneath the top that represent the eight octants of cartesian three-space (like the quadrants of two-space). If the object doesn't occupy space in an octant, then that node terminates; otherwise, an octant is itself subdivided into octants, and the process continuues to some determined resolution. This method is also primarily used in mechanical engineering for the same reasons.

The third method is the most common in non-engineering applications. This method is called boundary representation, or B-rep. Objects are represented with polygonal surface approximations. Surfaces are typically stored as lists of vertices and attributes, such as color, and surface normal, in tables. These are accurate for objects with flat surfaces, but not for curved surfaces, which is why they are not good for engineering analysis. Complicated curved surfaces can be stored as Coon's patches (or with other techniques), but working with and displaying this type of graphic information is slow and difficult. Polygonal approximations can be rendered, using Gouraud or Phong shading algorithms, as smooth, rounded surfaces.

The three common methods of representing graphic information are not mutually exclusive. CSG techniques for data representation are often used to generate data that is then stored as B-rep. Storing a low-resolution copy of a B-rep description in an octree has been shown to give enormous speed improvements in some computation intensive algorithms such as ray-tracing.

6.2.2 Procedural Knowledge

Procedural knowledge is represented in functions or procedures. There are several advantages to this technique, also. It is much easier to encode repetitive or ordered information such as a high rise elevation, and it is much easier to encode complicated but regular descriptions such as a sphere as a point, a radius, and the rules for generating the points on the surface. (They are calculated from this description, rather than trying to store them all.) Since the knowledge is an integral part of the program, it is difficult to modify or extend, and it is also difficult to work with just a part of the information.

There are two methods of encoding procedural information: as rules, and as relations.

Rule based procedural definitions of knowledge come in two types. The simplest is the cookbook type. The knowledge is in a list of steps that define the process that defines the information. Recipes are fairly easy to write, when the steps are understood, but they define the same constant form every time. The second type is an extension of the first that includes conditional rules. As the recipe is being followed, the next action depends on the current state or configuration. Generative grammers and production systems are both based on conditional rules. It is fairly easy to write conditional rules that will generate many very different solutions depending on the arguments or parameters the recipe starts with. This is partly due to the fact that new configurations are synthesized.

The other method of encoding procedural information is through relationships. A recipe is again written using a few key variables, that are specified as arguments or parameters, that in turn define the object. This is analogous to the computer programs written using the ubiquitous personal computer spreadsheet. As the recipe is being followed, the size, position, angle, color, etc. of the next element is defined in terms of the other elements; the first ones are defined in terms of the parameters. This method is good for sensitivity studies where an element is changed to discover its effect on the rest. William Mitchell calls this "parametric variation."

6.2.3 Hybrid Representations

There are two ways to get many of the advantages of both representations. (Both methods assume that the stored representation is declarative.) A procedural description can be executed and the generated information can be stored declaratively. The only disadvantage to this technique is the trade-off between accuracy and space. For example, a procedurally defined circle is very accurate, but the declaratively defined circle is a polygonal approximation. More space in computer memory is required for a more accurate approximation. It is also possible to write commands into the system command language that operate on the declarative information and extend or modify the object description. Commands to repeat and extrude are examples of this. This technique can be taken further, allowing the designer to write programs to generate information. This will be discussed in the next chapter.

6.2.4 Abstraction

Means of systematizing knowledge about a project become very important when the knowledge is either extensive or complicated.

Any project one might want to represent can be considered to be an assembly of parts. If the model is of the visual characteristics of an elevation, the parts might be the building outline, the doors and the windows; or they might just be lines and polygons. If the model is to be used for financial or energy calculations, attributes of the elements become important.

A creatrix must have the ability to collect primitives together into objects. In drafting systems these collections are often called symbols. To use objects in design, we need to be able to ascribe attributes to each object so that they are not just collections of lines and polygons.

Describing models by describing their parts is a way of using hierarchical abstraction. By allowing objects to contain other objects a creatrix will be able to completely represent this type of structuring. The object/subobject structure doesn't have to mirror the physical structure. It might represent general uses of spaces and more specific uses of those spaces, for example.

6.3 Second Order Architectural Knowledge

Second order knowledge expresses knowledge about the interrelationships of objects and the relationships of objects to the world. This can be expressed as requirements in a configuration. They are often called constraints.

Many relationships are a direct expression of the client's program, and are constraints on the design of a project. As long as they are satisfied, their explicit representation is redundant because the plan implicitly encodes the information. But as changes are made to a scheme, they might become unsatisfied.

Usually an analysis of some type of architectural model requires knowledge of the inter-connections of the parts in terms of relationships between them. (Visual analysis might seem to be an exception, but actually it requires specific spatial relationship information. It is just that this spatial information is inherent in the geometric representation.) Many operations on the information can only be performed when relationships are known. These connections can be either physical (the kitchen is next to the dining room, a black stone floor absorbs heat from the sunlight falling on it) or semiological (the hearth should be a symbolic center of the house; there is a peaceful passage from the lobby to the auditorium). The first are much easier to deal with.

A creatrix could be built that would be able to perform calculations requiring intelligence. To do this it would need to know what a kitchen is and what a dining room is to determine if they are next to each other or not. This is feasible with many physical relationships, but becomes difficult with the conceptual relationships. How should a creatrix determine whether or not the hearth is a symbolic center of the house? The construction material of a seemingly unrelated wall might change the hearth's symbolism.

When modeling for physical performance analysis, the primitives depend on what needs to be represented for the analysis, and also on which of often many techniques is used for the analysis. To calculate floor areas automatically, the creatrix needs to recognize walls and doors. To calculate rentable space or circulation efficiency, it needs to understand halls, vestibules, shaft space, etc. This automatic calculation is unreasonable when the architect is unsure of these details himself.

Chapter 7

Operations on Concepts

Architects normally interact with their ideas through paper and pencil. It takes little skill to make a graphic expression, partly because we are so used to sketching. People learn to draw before they learn to write. Only when drawings become inadequate in expressing ideas does a designer turns to scale models. Drawings and scale models are the same thing when the representation is in a creatrix. There are other ideas that need to be expressed about a project that are often left out of the drawings, but can be included with a creatrix, such as required mininum and maximum room dimensions. These ideas are often expressed more easily with words or equations than with drawings. A designer using a creatrix needs to express essentailly graphic ideas, but also verbal ideas.

This chapter loosly describes the commands that need to be available in a creatrix for the expression and manipulation of concepts.

7.1 Command Schemes

7.1.1 Interrogation

The simplest "user interface" is the interrogation type. The user types the name of a command or computer program, and the program then proceedes to ask for each piece of information that it needs as it needs it. For example:

```
> sunangle
What is the latitude in degrees? 46
What is the month? March
What day of the month? 21
What time of day (24 hour clock)? 12:00
The azimuth is 180 degrees and the altitude is 44 degrees.
>
```

It is very easy to use such a program or command. It takes no special skills to use and requires little or no time to learn. This scheme is best in situations where there are only a few facts needed, and when it is difficult to remember the details needed by the program.

7.1.2 Menu

A more sophisticated user interface is the menu. The user picks the operation wanted from a displayed list of posible choices. Some graphics systems use a menu with hundreds of commands on one large menu. Instead of one menu item for "insert line" and another for "insert polygon", a hierarchical system of menus can be used. After "insert" is choosen, there would be another level with the choices "line" and "polygon." This is called a dynamic menu; it alters the possibilities for another selection depending on the last selection. (Most spreadsheet programs use a dynamic menu.) Menu items are usually selected by pointing at them, either with a stylus on a Rand tablet or with a mouse. Straight menus are usually considered to be inefficient by experienced users. Dynamic menus are much better, and they are best when there are only a few options at each level.

Menus can be very easy to use because all the possibilities are visible at once. beginners appreciate the simplicity of the menu, all options are visible to choose from and there is no syntax to learn.

7.1.3 Command Language

A dynamic menu can be so flexible that it approaches the third command interaction scheme. It is called a command language or "dialogue" and consists of a problem oriented language (POL). It usually takes much longer to learn to use a command language than a menu. A user needs to remember not only the words and their meanings, but also what combinations of them are possible.

Command languages are much more powerful than menus when performing anything but very simple operations. This is because the exact command can be "built" from the vocabulary, and complicated commands can be constructed from many simple commands. I know of no one who uses a menu when a command language is available, even the non-typists, except for beginners.

7.1.4 Natural Language

Natural language is very difficult for computers to understand. Humans combine an understanding of the context with the meaning of the sentences in order to resolve possible ambiguity and understand the communication. Computer systems that are capable of some understanding of natural language have been built, but they are large and slow. When the vocabulary is limited, it becomes much easier to include natural language understanding in another program. Some successes have been made with limited vacabulary systems. If restrictions are put on the syntax also, it becomes much easier to recognize natural language, but the language is becoming less natural. This is not necessarily bad, English[10] does not express mathematical ideas as well as algebraic notation.

A creatrix should be able to understand simple commands expressed as

[10] or Danish, or Portuguese, i.e. any natural language

natural language, but she should translate them into a target language that more advanced users would use for efficiency and accuracy. Ambiguities would be resolved with interrogation. Errors would be undone with an "I didn't mean to do that." command.

Voice recognition is another difficult task. At this point, they still require a user to talk in a stilted way, and require frequent repetition. As the technology becomes better, they can be used with a creatrix to replace the keyboard.

7.2 System Operations

7.2.1 Viewing and Drawing

Architects use orthographic projections because they can easily be drawn to scale, and can easily be measured. Perspective sketches are common, but measured perspectives are usually only done for presentation. The creatrix makes perspectives as easy to produce as plans and elevations, but all are needed.

For the parallel projections, angles of rotation are needed which determine the type of projection. An image size also needs to be specified.

Perspectives are more complicated. The eye point or station point needs to be located in 3-space. The direction of view is also needed. It can be defined as heading, pitch, and roll; or as an object point (a point on the "optical" axis) and an indication of which way is up in the image. In order to be able to fully control the perspective image, synthetic rises and shifts of the view camera[11] are needed. This allows perspective images of tall buildings to stand up straight.

When the scheme is to be displayed interactively, it can be shown as either

[11]It is suprising that 3-D graphics systems do not all have this capability since it is so easy to provide.

lines or surfaces. Lines that would be hidden in reality can be shown or hidden. Removing the hidden lines is a slow process, and has to be done off line.

If it is going to be plotted, a drawing size is needed. This might be an overall size of the drawing, or for parallel projections it might be expressed as an architectural or engineering scale (like 1/4" = 1'-0").

When the scheme is rendered as surfaces, they need to be drawn correctly, or the image makes little sense at all. Depth sorting is fast enough for interactive use, but is not always accurate, other techniques are too slow without special hardware. More accurate methods can be used off line, and should be available.

Surface images need to be shaded. Lambert's cosine law for surface shading with a light source at a specified location is needed. More sophisticated shading techniques would be part of the hidden surface program.

7.2.2 Construction Tools

The fundamental tool for graphic data construction is the coordinate system. It provides a method to locate points in space. The most common is the rectilinear Cartesian system. It is ideal for describing objects with right angles. There are other schemes that describe locations with angles cylindrical, polar, and revolute). The first is ideal for describing objects basically rotated about an axis, such as round towers. The second for objects with spherical forms, such as domes. surfaces. The third is good for describing things like the intersection of groin vaults. The creatrix stores location data in Cartesian space, but the designer should be able to interact in any system.

Snapping is what the creatrix does when it takes a point and moves it to a nearby location determined by some simple spatial constraint. There are several types of snaps possible. One is to round the coordinates to the nearest tenth, thousandth or what ever. Another is to move the point's location to the location of the nearest point. Points can be snapped to predefined collections of locations called grids. The constraints can be applied to the objects the points define. A point can be snapped so that a line will be some length. Two lines can be made to form some angle or be parallel. One or more points can be adjusted so that a polygon can be made flat, or regular.

Grids can be regular or tartan, and they can be planer or 3-D. The architect should be able to define the grid as a plane (cartesian or polar), a cone, a cylinder, or a sphere (geodesic or latitude and longitude).

If the display hardware is capable, rubberbanding and dragging are useful, but not essential.

Coordinate schemes always have an origin where the three variables are all zero. There are situations when a different origin and rotated axes are very usefull, such as when adding detail to a sloping or angled wall. The specially defined coordinate system is called the working axes. It can use any of the coordinate systems.

7.2.3 Programmability

A lesson which many have learned the hard way (and more have never learned) is that it is not possible to provide all the commands that might be needed when the task is undefined. The best that can be done is to provide all the basic commands that might be needed, and then provide an efficient means for the architect to define his own commands. This means the creatrix must be programmable.

The programming language should be interpreted so the architect can build his programs incrementally. The language should be standard so that existing material can be used for training. The language should be easy to use, but capable of serious work. LISP, Forth, and Prolog are all candidates. LISP is probably the best because of its strong capabilities and proven record.

7.2.4 Miscellaneous

Just as different working axes are usefull, so are different units of measurement. So that a designer can work on the plan in feet, and sketch a detail in inches, the units can be changed.

There are many "default" values to system variables that need to be monitored and modified such as the current set and label that information being added is associated with. Commands are needed for these tasks.

Commands are needed for file management: loading in files, saving files on disk, scanning files for contents and so on.

To allow for the graceful recovery from mistakes and mind changes, memory of the last dozen commands will be saved so that they can be undone one at a time until the last correct stage is reached. Like some word processors, the system could save the representation to disk files every twenty commands so that more major errors could be undone.

Because the program will have more capabilities than someone can learn right away, a system for online help will be available. The architect would be able to ask what a command does and what command does what he wants. A good help system is not a substitute for an easy to use command scheme.

7.3 Data Operations

7.3.1 Point Location

The most basic aspect of graphics commands is the ability to specify locations somewhere in space. Lines and polygons are defined in terms of locations; lengths and areas are calculated in terms of them. The most common scheme is to represent points as coordinates in Cartesian three-space, expressing locations as distances from an origin along three axes. It is easy to understand and to work with for most spatial problems. The other schemes are spherical, cylindrical, and revolute; all involve angles. These systems can make some problems much easier to work with, and it should be possible to specify locations in these ways also.

Some sort of cursor or crosshairs on the screen is needed to point with. This might be controlled with a Rand tablet's stylus or puck, a mouse, a light pen, cursor keys, or a joy stick; they all have their advantages. The Rand tablet is needed to digitize site plans and other existing graphic images. The other devices are cheaper.

A designer often thinks in terms of approximate sizes, without specific dimensions. A designer needs to be able to specify a location by pointing at it.[12] When he wants to work with measured drawings, pointing alone is insufficient. The simplest technique is to specify values for the X, Y, and Z coordinates. It is usually not very convenient, but is sometimes needed.

A location can be specified relative to an existing point (or points). There are several ways to do this. A point can be defined relative to another point by distance along one or more of the axes. A point can be defined as the intersection of two

[12]With two-dimensional display devices, a location can only be pointed at relative to some plane.

lines, or the intersection of a line and a plane. A point can be defined as lying some ratio of the distance between two other points. A point can be defined as one of two that are equidistant from two other points. A point can be defined as the location that is equidistant from three other points. Because some of these methods refer to existing points, all these methods of selection of a point must be avaiable (see the next section).

There are many other possible ways of specifying points that are more difficult to understand and use. A point can be defined as the location between two other points that would form a perpendicular intersection with a third point. A point can be defined as the mean (arithmetic or geometric), median, or mode[13] of a set of points.

It must be possible to combine the X, Y and Z components of points defined separately as above to form a new point. This would allow a designer to define a point that has an X value that is the same as the X of the intersection of two lines, and has Y and Z values that are offset from some point. It is important when a point is specified by refrencing other points, that the other points can be specified in any of the ways for specifing locations.

7.3.2 Data Selection

The most basic aspect of commands in general is the ability to select the data or information that has already been defined in order to change it, make calculations or inferences with it, or eliminate it.

If the data to be selected is graphic or has a graphic representation, then there are several methods possible. You can simply point at it. Usually there is a

[13]Remember that the points might have an abstract meaning, not necessarlly a spatial representation.

density of information that requires the designer to specify what type of data he wants to select: point, line, object, node, etc. The closest instance (or instances if several were picked) is returned as the selected data. Or you can define a perimeter on some plane and ask for everything either inside or outside of the polygon (extruded along some projection, usually the principle axes). Or you can define a volume for either its contents or the excluded data. Or you can define a constraint such as "within 5 meters of this point", or "above this point."

If the data is not represented graphically, or if it is not convenient to select it graphically, then it can be selected by describing its attributes. These might be simple atom characteristics such as membership in a set or group, pen number, color description, font name, etc. Or they might be complicated attributes such as "polygons with perimeters greater than their closest distance to the origin." This type of command is very usefull, and will make a creatrix much more powerful. This capability is important when dealing with ideas, and not just drawings.

When selection is in reference to other data, all the techniques for selection must be available. In some circumstances, it is important to be able to select data that doesn't exist, so the commands for adding information must be available. When selection involves locations, all the techniques for specifing locations must be available.

7.3.3 Basic Operations

The first command needed is to add information to the representation. The simplest form would be to add a node, a line, a polygon etc. This fits well with ideas of drafting, but not of architectural design, unless the command is so easy that no thought is given. This can be the case when working with a Rand tablet, but it is 2-D. The working axes can help locate data in 3-D, but it is still 2-D and any one time.

Sketch recognition is required for creative 3-D work. The system has to be good enough that it doesn't make mistakes of interpretation. The creatrix will be able to understand sketch input using a new technique [Jurgensen 86b]. This aspect of the creatrix is of fundamental importance. Here is where fresh ideas flow from the mind, and the river must be straight and deep.

N. John Habraken has developed a notation [Habraken 83] for describing built form that is ideally suited for input to a creatrix. It is called "writing form." Its basic data elements are lines and distances. These are combined with basic moves: *string* to make a horizontal linear assembly; *stack* to make a vertical linear assembly; *replace* to substitute one element with another. There are transfigurations of magnification and reduction. There are also rules that relate elements to each other. For some types of building description it would be very good, but I don't think most architects think of their schemes as strings and stacks.

The most basic information modifing commands are: move or translate, turn or rotate, and size or scale. These need to be applicable to and data atom or selection. If a point is used to define two polygons, there are times when moving the point should move the vertex for both polys, and others when it should only move one. A way of controlling this is needed.

When shemes change, information must be removed from the representation. A delete command that can operate on anything selected is needed. The deleted information would be restorable up to twelve operations back.

A replace command will allow one collection of information to be substituted for another. This should work with the select.

A swap command would take two selections and substitute one for the other.

A reflected or mirrored thing is a reversed and moved copy. It is equivalent to scaling by negative one across some plane.

The ability to skew or scale things proportional to their distance from some location is helpful.

Because lines and polygons are represented as ordered collections of points, a means of inserting a new point into the sequence is needed. This subdivides a line or polygon edge. A means of specifing where is needed.

Many commands are needed for working with the structure of the representation. Object descriptions are hierarchical, and are assembled in a network or lattice. Commands for pruning, grafting, growing, and transplanting are needed.

Many basic forms are most easily described by defining relationships of their parts. These relationships are called constraints. By using constraints, a designer can model complicated ideas, and change aspects while the creatrix changes the things that were affected. This is called propagating constraints. When the definition is under or over constrained, special techniques are needed. Geometric constraints can be expressed as numeric constraints on point coordinates, and solved by numerical methods (such as Newton-Raphson iteration and relaxation). The basic constraints for graphic representations are: freeze location, distance, direction, angle; equalize x value, y value, and z value; equalize distance, direction, angle; minimize distance, direction angle; maximize distance, direction, angle. SKETCHPAD was a constraint system [Sutherland 1963]. Another two-dimensional constraint system is discussed in [Nelson 1985].

Because the creatrix represents things as collections of fundamental atoms, there are circumstances where a designer either wants to change the representation means without changing the basic effect (for example: change a polygon to a set of lines), or wants to apply one type of atom to another (change a string of lines into a sequence of eye points). This translation capability allows operations that only make sense for one atom type to be applied to another (lines do not have area), and it obviates the need for many esoteric commands.

7.3.4 Information Expansion

Many of the efficiencies of a procedural description can be simulated with operations on declarative information. The basic operations are repetition of a copy of some thing translated, rotated or scaled one or more times. This lets one create rows and rings of things. Adding the capability to connect the original with the copy (with the appropriate atom type) provides several types of extrusion and surfaces of revolution. Rather than limit the copies to translation, rotation, and scaling, an arbitrary path should be possible. Many very irregular forms would be easy to model.

In some circumstances, it is good to have a terrain to "build" on. Fractal techniques allow a simple terrain approximation to be rendered in more detail.

Curve fitting techniques allow a complicated form to be defined by control points which are interpolated to describe a curve or curved surface.

For imaging purposes, polygons should be displayable with synthetic textures such as wood grain, brick, stucco, etc. This increases the information content in the image without requiring complicated modeling.

7.3.5 Calculations

Because a creatrix stores information about a scheme and not just a delineation of the form, many calculations can be automated. Some of the simpler technical calculations can be made on more structured descriptions, calculations from graph theory can be made on networks, and many geometric calculations can be made directly,

When the representation is structured to represent physical objects, some simple technical calculations can be made. A creatrix only needs to work with rules of thumb for approximation, accurate calculations are too involved. Detailed studies are best left to specialized computer programs.

Daylighting requires knowledge of opaque and transparent surfaces, as well as sun positions. Artificial lighting calculations need descriptions of the light sources. Acoustics calculations need surface characteristics. If thermal conductivity characteristics are known for all surfaces, simple energy use calculations can be made. When connectivity of structural members, their materials, and loads are known, simple structural calculations can be made.

The creatrix is capable of representing systems as networks of nodes connected by arcs. There are many problems that lend themselves to efficient solution when represented as a network using techniques from graph theory.

When problems are complex, decomposition can help as an intelligent divide and conquer strategy. Flow calculations can be used in circulation analysis. Other problems can be solved using shortest path, spanning tree, and other techniques. This representation can help with room layout. With two or more points, calculations can be made for: distance or length; difference in x, y, and z; and perimeter. With three or more co-planar points, or a polygon, calculations can be made for area. With four or more non-planar points, or four or more polygons, calculations can be made for surface area and volume.

Two or more points, two or more lines, and two or more polygons can be checked for coincidence. Three or more points, two or more lines, and two or more sides of a polygon can be checked for colinearity. Four or more points, the points in one polygon, and two or more polygons can be checked for coplanarity.

All the basic calculations of angles, tangents, intersections, perpendiculars, normals and parallels are needed for all applicable atom types. There need to be both commands to find the angles and etc., and also inverse functions that produce (for example) a line that has some angle to another line.

With two polygons or two volumes, the Euler operations of union, intersection, and difference calculate new polys or volumes that are different combinations of the two original.

When an architect has a photograph of a site, a creatrix can calculate the camera location so that a perspective can be made of the creatrix's representation that matches the photograph.

Because if the fundamental importance of sun-light and shadows in architecture, a creatrix needs to be able to represent shadows. The calculation of the position of the sun for a given latitude, date and time should find a point. Shadows can be cast for a light source from any point. Optionally, calculations for the planets' positions could be included in a creatrix, and the starving artist could help to pay for his computer by selling astrological services.

Chapter 8

Conclusion

The creatrix has been specified to be an instrument for the conceptual design of form. Because conceptual design involves abstract concepts rather than just physical models, a creatrix should (theoretically) be able to operate not only for architectural design, but also textile design, industrial design, graphic design, furniture design, sculpture, and so on. Conceptual design of any two or three dimensional objects with a creatrix should be possible. There will be problems using a creatrix for the conceptual design of form that changes in time, such as in music, dance, and story telling.

The first generation of creatrixes will work with concepts of static states - which should not be confused with dynamic interaction with the concepts such as movement through space.

After the first creatrix is built, it could be used to design a second generation creatrix, which could be used for the third, either ad infinitum or until they started designing themselves. Of course it will soon be solving the worlds problems, or at least to design ones. Maybe I can sell you some vaporware?

References

[Abelson 85]	Abelson, Harold and Gerald Sussman with Julie Sussman. *Structure and Interpretation of Computer Programs.* MIT Press, 1985.
[Akin 82]	Akin, Omer; Eleanor Weinel. *Representation and Architecture.* Information Dynamics, Inc., 1982. collection.
[Albers 79]	Albers, Anni. *On Designing.* Wesleyan University Press, 1979.
[Alexander 64]	Alexander, Christopher. *Notes on the Synthesis of Form.* Harvard University Press, 1964.
[Banham 74]	Banham, Reyner (editor). *The Aspen Papers.* Praeger, 1974. 1974 overview of twenty years of conferences.
[Begg 84]	Begg, Vivienne. *Developing Expert CAD Systems.* Unipub, 1984.
[Brodie 82]	Brodie, Michael; John Mylopoulos; Joachim Schmidt (editor). *On Conceptual Modelling.* Springer-Verlag, 1982. 1982 Seminar.
[Charniak 85]	Charniak, Eugene and Drew McDermott. *Introduction to Artificial Intelligence.* Addison-Wesley, 1985.
[Coons 63]	Coons, Steven Anson. *An Outline of the Requirements for a Computer-Aided Design System.* Technical Report, Massachusetts Institute of Technology, 1963.
[Eastman 75]	Eastman, Charles (editor). *Spatial Synthesis in Computer-Aided Building Design.* Halsted Press, 1975.

[Gross 86] Gross, Mark.
Design as Exploring Constraints.
PhD thesis, MIT, February, 1986.

[Habraken 83] Habraken, N. John.
Writing Form.
1983.

[Hawkes 75] Hawkes, Dean (editor).
Models and Systems in Architecture and Building.
The Construction Press Ltd, 1975.
LUBFS Conference number 2.

[Johnson 70] Johnson, Timothy; et.al.
IMAGE.
Department of Architecture, M.I.T., 1970.

[Jurgensen 85] Jurgensen, Peter.
What are Architectural Design Tools?
June, 1985.
from Don Schon's design research seminar at MIT on June 1, 1985.

[Jurgensen 86a] Jurgensen, Peter.
Sketch Recognition.
1986.
unpublished notes.

[Jurgensen 86b] Jurgensen, Peter.
A Knowledge Representation Scheme for Design Support Systems.
Technical Report, Massachusetts Institute of Technology, 1986.
School of Architecture and Planning--Computer Resource Lab.

[Kennedy 71] Kennedy, Michael (editor).
Kentucky Workshop on Computer Applications to Environmental Design.
University of Kentucky, 1971.

[Koffka 35] Koffka, K.
Principles of Gestalt Psychology.
Harcourt, Brace and Company, 1935.

[March 76] March, Lionel.
The Architecture of Form.
Cambridge University Press, 1976.
collection.

[Moore 70] Moore, Gary (editor).
Emerging Methods in Environmental Design and Planning.
MIT Press, 1970.
1968 conference of The Design Methods Group.

[Nelson 85] Nelson, Greg.
Juno, a constraint-based graphics system.
In *Computer Graphics.* ACM SIGGRAPH, 1985.

[Porter 69] William Porter.
The Development of Discourse: A Language for Computer Assisted City Design.
PhD thesis, MIT, August, 1969.

[Ross 60] Ross, Douglas.
Computer-Aided Design: A Statement of Objectives.
Technical Report, Massachusetts Institute of Technology, 1960.
An excellent paper.

[Ross 63] Ross, Douglas and Jorge Rodriguez.
Theoretical Foundations for the Computer-Aided Design System.
In *Spring Joint Computer Conference Proceedings.* AFIPS, 1963.

[Ross 64] Ross, Douglas and Clarence Feldman.
Verbal and Graphical Language for the AED System: A Progress Report.
Technical Report, Massachusetts Institute of Technology, 1964.

[Schon 63] Donald Schon.
Displacement of Concepts.
Tavistock Publications, 1963.

[Schon 83] Donald Schon.
The Reflective Practitioner.
Basic Books, 1983.

[Sowa 84] Sowa, J. F.
Conceptual Structures.
Addison-Wesley, 1984.

[Stoker 86] Stoker, Douglas.
personal conversation.
1986

[Sutherland 63] Ivan Sutherland.
Sketchpad, A Man-Machine Graphical Communication System.
PhD thesis, MIT, January, 1963.

[Unknown 72]	Unknown (editor). *International Conference on Computers in Architecture.* Unknown, 1972. University of York, September 1972.
[Waddington 77]	Waddington, C.H. *Tools for Thought.* Paladin, 1977.
[Werteimer 59]	Michael Wertheimer. *Productive Thinking.* Harper & Row, 1959.