The Shape of Digital Content

A Computing Language Based on Gibson's
Ecological Approach to Visual Perception
and the Theory of Shape Grammars

by

Mark M. Watabe

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 2010

Signature of Author: _____
Department of Architecture
May 20, 2010

Certified by: _____
George Stiny, PhD
Professor of Design and Computation
Thesis Supervisor

Accepted by: _____
Julian Beinart, BArch, MCP, MArch
Professor of Architecture
Chairman, Department Committee on Graduate Students

John Ochsendorf, PhD
Associate Professor of Building Technology
Thesis Reader

Vincent Lépinay, PhD
Assistant Professor of Science, Technology, and Society
Thesis Reader

The Shape of Digital Content

A Computing Language Based on Gibson's
Ecological Approach to Visual Perception
and the Theory of Shape Grammars

by

Mark M. Watabe

Submitted to the Department of Architecture
on May 20, 2010 in Partial Fulfillment of the Requirements
for the Degree of Master of Science in Architecture Studies

ABSTRACT

James J Gibson, a psychologist who wrote prolifically about his theories of visual perception argued there is a difference between direct information, that is the perception of the affordances in an environment, and indirect information, that is the information contained in signals and signs in a defined channel of communication. This thesis develops a framework based on this distinction proposed by Gibson to show how networked electronic devices are used both to communicate and to enable new modes of perceiving the environment. Based on my analysis I will argue that the framers of what is called "The Semantic Web" are primarily concerned with enabling communication of indirect information and this is preventing the development of more innovative interfaces that enable networked humans to directly perceive and interact with their environment in novel ways. I will then explain an alternative framework for describing what might called the "The Semantic Web" by relating Gibson's theories to the more formal theory of Shape Grammars developed by Professor of Design and Computation George Stiny. Stiny's work illustrates how the semantics of direct information, that is interacting with shapes as shapes, are very different from the semantics of indirect information, that is interacting with shapes as symbols. I will then describe a software demo I developed based on these arguments and explain how it is different from the Semantic Web as currently understood.

Thesis supervisor: George Stiny
Title: Professor of Design and Computation

Acknowledgements

Thank you to the rest of the computation group faculty and my fellow peers. Special thanks to my advisor George Stiny. In my two years here I have heard your presentation introducing Shape Grammars no less than six times, and up until the very end I still was unsure what to make of it. It wasn't until my third semester, when I took a variety of classes in other departments, that I saw the value of the whole shape enterprise and realized these concepts will always be inseparable from my own life project.

Thank you Tiffany for always bringing me to the here and now.

And thank you to my family for believing in me and always supporting me. If I am blessed with any intellectual curiosity it is because of all you have done for me.

Contents

Perhaps the composition and layout of surfaces constitute what they afford. If so, to perceive them is to perceive what they afford. This is a radical hypothesis, for it implies that the "values" and "meanings" of things in the environment can be directly perceived. Moreover, it would explain the sense in which values and meanings are external to the perceiver.

James J. Gibson

There's an imbalance today --rote counting passes for thinking far too often, with preposterous results. Everything is numbers and symbols without shapes. Seeing doesn't need to be explained, it needs to be used.

George Stiny

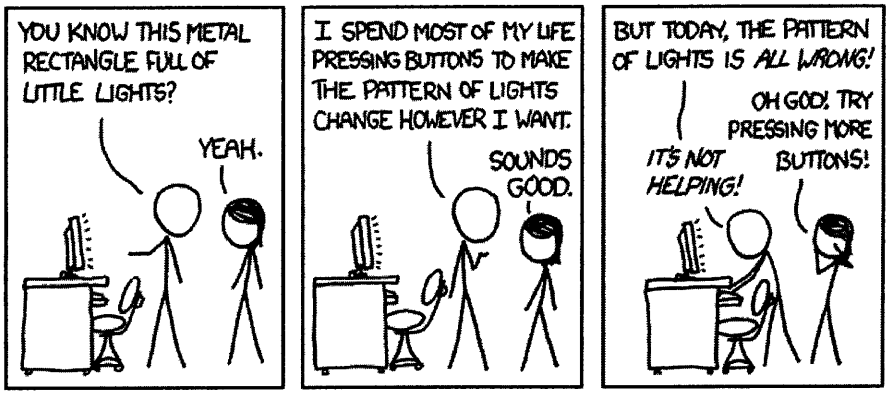


Fig 0.1 [F-1]

1. Introduction

1.1 Introduction

What do we tell ourselves we are doing when we wave our hands and stare at a screen? Both "changing the current state of bits" and "changing the color of pixels" are not very useful descriptions as they apply to every activity we do whether we tell ourselves we are "drawing a building", "writing a thesis", or "slaying ogres". But what each of those activities does have in common is that they are performed within some interface that was *coded* by someone or some collection of people. We currently understand "coding" to be the process of specifying combinations of symbols, as primitive data types of numbers, characters and booleans. But what if code itself were composed of shapes themselves, that is points, lines, and planes, instead of numbers and characters that must be represented graphically, that is *as shapes*. And what if the parts we understand ourselves to be acting on when we are interacting with our screens are these same shapes? Wouldn't this transform computing from an abstract process (whose full extent is never known) that needs to be visualized with a spatial metaphor, into *the* process we tell ourselves we are looking at when we stare at our screens? Wouldn't this make us all "coders"?

My argument for why such a language, both as code and as the interface, is possible and worthy of investigation is based on the work of two theorists, the late psychologist James J. Gibson and my own advisor, Professor of Design and Computation at MIT, George Stiny. Gibson's arguments will supply the *why,* and Stiny's arguments will supply the *how* for this investigation. I will use Gibson's terms and concepts described in his text The Ecological Approach to Visual Perception to analyze the short comings in the discussion about how best to use networked electronic devices in society, and show how this assumption that *appearances* can be separated from *information* has prevented thinking about how to collapse the gap between how users specify machine readable information and human readable information. Stiny's theory of shape calculation, or shape grammars, as expressed in his 2006 text Shape will provide the technical foundation for how a programming language based on shapes, or what he calls maximal parts could be designed. Both Stiny and Gibson reject the atomistic reductionism of the natural sciences and attempt to formalize how one can describe visual reality, it's parts and how it changes, without needing to make claims about *what things are*. These epistemological concerns only prevent an individual from seeing what he or she can *do*.

1.2. Clarifying what we mean by "appearances", "information" and "meaning"

Listening to the way that scholars and technologists have framed the debate about how we ought to use networked electronic devices in society it is obvious they are uninterested in considering how the appearance of the monitor fits in with their concepts of information, code, speech, meaning, etc. In the foreword to Spinning the Semantic Web Tim Berners-Lee, who is credited with inventing the internet, laments the fact that development of what he refers to as the Semantic Web, *"a web of machine-readable information whose meaning is well-defined by standards"* has *"taken a back seat to the Web of multimedia human-readable material."* (xi) The markup base technologies supported by the authors of the book treat the composition of the monitor as an arbitrary image that is the result of styling that is applied to a source file of well defined information. But if human's act based on what they see, and machines act according to the strict logic of how the code is defined, then isn't the opposite approach the only way to make these two layers converge? Should it not be the case that machines only specify what it is that humans *see* and do not treat the "multimedia human-readable material" as a separate layer from the underlying logic of the machine? This approach would of course require that we believe human's can see the "meaning" of appearances without being told what they *ought* to mean.

Gibson does believe that this is possible and asserts that there are two distinct types of information, and therefore two distinct types of meaning. What we traditionally call information Gibson qualifies as being *indirect information,* that is the information contained within human constructed channels of communication. [Gibson, 1986 p. 62-63] In communication meaning is a human construct that requires at least two people agree on what a certain signal or symbol indicates. But seeing the human assigned meaning of specific forms in what Gibson calls, *"the flowing optic array",* that is the appearance of the environment as we move around it, is different from directly perceiving the *affordances* of the surfaces we see directly in the array. When an animal sees the affordances of the environment it sees potential actions it can take or potential actions that might happen to its being. In the case of direct perception the *meaning* of the surfaces an animal sees are specific to that animal. Whether the perception

of a smooth surface affords lying on depends on the capabilities and form of that animal in relation to that surface. And whereas seeing the meaning in symbols requires an individual be socialized, animals learn the meaning of surfaces through ongoing interaction with the environment. [Gibson, 1986]

Considering Gibson's distinction between these two types of information leads us to ask: Does the composition of the screen reveal the output of a defined channel of communication, or does the screen reveal a new technology enabled view of the affordances of the environment that span durations and extensions before unseen? The screen reveals both of course, but because we understand the internet to be a tool for indirect communication, we are ill equipped to discuss the full effects and potentials of so many networked devices that are programmed to specify combinations of pixel colors by computing the actions of humans in relation to the current state of machines and sensors spread throughout the globe. When we use computer interfaces that are designed to be simulations of well understand physical processes, such as email as a virtual version of sending a letter through a physical network of humans and vehicles, we become blind to the reality of what we are actually doing and the other processes that are enabling the simulation. When I send a friend a physical letter it does not arrive with ads based on the content of what I have written as my emails in G-mail do. And I am not simply trying to frame a discussion based on notions of "advertising", "privacy", "surveillance", etc. but instead propose that if we can close the gap between graphic parts and computer descriptions we can develop new terms to talk about the design of digital environments. All acts taken for the purposes of indirectly communicating with someone else through a defined channel of communication are still acts in the world, and from a third perspective those acts do reveal affordances in the environment. Hence any composition that is the result of an action in a networked digital context is not simply a speech act that may or may not be private, but is a mediation between being an act of communication and an act to be observed by people and machines.

1.3. Problem statement

The aim of this thesis is to develop new terms for discussing digital environments based on relating James Gibson and George Stiny's work. The current terms and techniques for computer human interaction are littered with vestigial metaphors leftover from specific moments when physical processes gave way to digital ones. Web content is still based on markup text and pages, and the operating system is still represented as files and folders. But these metaphors often disguise what is going on "behind the scenes". If graphic parts are acted on directly both as a user interface and as code, then perhaps it will be easier for all users, and not just coders, to understand what it is they are doing.

2. Relating Stiny & Gibson

2.1. Introduction

Before arriving at MIT for my two year SMArchS degree I was unaware of the work of Stiny and Gibson. I was introduced to the theory of shape grammars and visual calculating as described by Stiny in my first semester, and I was introduced to the work of James J. Gibson in two departments at MIT, (neuroscience and Science, Technology and Society) during my third semester. As I have come to understand both texts it is impossible for me now not to interpret one in relation to the other. Furthermore, I also cannot help but interpret their work as it relates to the design of digital interfaces even though neither theorist addresses that issue outright. But what is crystal clear in each author's text is the underlying incentive to describe perception without needing to claim what things are, and without reducing the world to theoretical parts that we pretend we understand. The world is out there to be seen, and what is most important is directly perceiving what we see in terms of what we want to do. Stiny expresses this thought through his simple equation, *"see x do y"* (15, Stiny) and Gibson makes similar points throughout his text and in his closing remarks, *"Perceivers are not aware of the dimensions of physics. They are aware of the dimensions of the information in the flowing array of stimulation that are relevant to their lives."* (306, Gibson) What Gibson and Stiny contribute is a way to talk about experience as direct reality. And by applying their work to the question of how networked electronic devices are used in society, hopefully we can begin to talk about the composition of the monitor as a part of the technical and political dialog, and not just as a problem for graphic designers.

2.2 What are the parts of reality

Gibson makes it clear that he is not interested in making claims about what the world "consists of", but instead is providing the minimum number of terms needed for describing the parts necessary for perception. This is an important distinction between how both Gibson and Stiny approach the way they describe reality from traditional science, they are not interested in classifying objects, or in reducing things to atomic units. Descriptions are only meaningful according to a specific instance or level of observation. In explaining the units of the environment Gibson says,

> *"for the terrestrial environment, there is no special proper unit in terms of which it can be analyzed once and for all. There are no atomic units of the world considered as an environment. Instead there are subordinate and superordinate units. The unit you choose for describing the environment depends on the level of the environment you choose to describe." (9)*

The examples Gibson gives of such units are mountains, canyons, trees, and cells. But aren't cells composed of something smaller? Gibson's way of describing reality, as we will see soon is much like Stiny's, because he provides terms that prevent unending atomic reductionism. He says the terrestrial environment can also be "*described in terms of a medium, substances, and the surfaces that separate them. (16)"* Many people often talk about imperceivable parts when talking about everyday life, such as "feelings are just chemicals in the brain" or "this book is really just a bunch of atoms". Gibson provides these terms to combat these cognitive leaps in understanding we make when talking about perception. When I look out

across a medium at blades of grass, the bark of trees, the shiny hood of a car, I am seeing the surfaces at the boundary of some substance whose composition is indicated *only* by its appearance. And although trees, blades of grass, and cars are identifiable objects that I can count, substances cannot be counted. Gibson says, *"It may be noted that objects are denumerable, they can be counted, whereas a substance is not denumerable" (34)*

Hence there are two kinds of decomposition in the way Gibson describes reality, either the identification of an object can be said to be composed of a subordinate unit, such as when one gets close enough to see the trees that make up the forest, or we can always resort to just saying an object consists of some substance. In either case we recognize that our calling out the perception of an object to begin with is only a matter of utility and not an epistemological claim. Stiny's concept of shapes, maximal parts and embedding is very similar in this respect.

> *Shapes are things like these [...] that I can draw on a page with points, lines, and planes. They're the first three kinds of basic elements of dimension zero, one and two. Shapes made up of basic elements have parts that can be seen in many ways. [...] Then, there are also shapes made up of solids. These are another kind of basic element of dimension three. Solids go together to make the shapes of everyday experience --the shape of things that are easy to hold and bump into. (2)*

Gibson is explicit that his notion of substance and surface have nothing to do with geometry, but there are still similarities between his terms and Stiny's concept of solids, 3 dimensional shapes, and planes, 2 dimensional shapes. There are two reasons why Stiny's definition of shapes are more aligned with Gibson's terms for substances and surfaces than with traditional geometry. One, shapes are always finite, and two, shapes of different dimension are not infinite collections of lower dimensional parts, that is a line is not composed of an infinite number of points. Lines are composed of lines and Planes are composed of planes. Gibson's definition of surfaces as the boundary of substances is identical to how Stiny defines the relationship between parts of different dimensions:

> *Basic elements that aren't points --either lines, planes, or solids --have boundaries. The boundary of any such basic element contains a finite number of basic elements, all of dimension one less than the original basic element itself. The boundary of every line is just two points. (171)*

Stiny's notion of a solid can be seen as formal description of the specific instance of a substance because he posits that the correct representation is the maximal element and not a combination of unseen components:

> *Maximal elements stand the customary atomic (analytic) view of things on its head. Every shape is defined crudely with discrete elements that are numerically distinct, but with no implied granularity or structure. The list of maximal elements contains the smallest number of biggest things that combine to form the shape, instead of the biggest number of smaller things that do. (183)*

What is interesting to me is that the importance of this concept of maximal parts, and its relation to Gibson, is easier to understand when we consider the computer representation of shapes. Current CAD

programs allow users to define geometry that might be composed of smaller than maximal parts. For example I can draw a line from the origin to (1,1), and a different line from (1,1) to (2,2) and so on extending in the upward diagonal direction to (10,10). Now suppose I leave and you come to look at the screen. You see one continuous line from (0,0) to (10,10). But how many lines are really "there"? Because computers must describe the underlying structure of appearances you would be unable to give the "correct" answer without guessing blindly. Judging by appearances there is 1 continuous line, but the underlying computer representation says there are 10 lines.

The theory of maximal parts is an important addition to Gibson's concept of substances and surfaces. When we think verbally we can look out at what appears to be a couch and say, "that is a couch", but I know I am using this concept to temporality stand in for "that is the surface of some substance". And when we think more geometrically we might look out and say, "that is a couch, but I know it is the maximal solid of the couch". The shape description says something about the current state of that substance and surface as it relates to me directly in the environment, the name description is only useful within a social context. "Will you carry my couch for me?"

At least for discussing the layout of surfaces in the environment, then, Stiny's concept of maximal parts provides one way of keeping descriptions in line with appearances. When a shape rule is applied overlapping maximal parts fuse to make sure that appearances conform to known descriptions. This will be important when we later consider how to make code based on shape descriptions as opposed to symbols that are represented as shapes. Symbols are saved in the computer as numbers, and numbers can be mapped to points on a line. Therefore code as symbols is saved from the problem that identical appearances might be composed of different parts. If Stiny's maximality is not enforced with shape descriptions then those parts can not be treated as code as there would be far too many possible computer descriptions for identically appearing parts. Such code would either take way too long to compile or produce unexpected results.

2.3. How the parts change

Gibson makes it clear that there are two very distinct types of change in the perception of reality, there is change to the substances and surfaces of the environment, what Gibson calls *ecological events*, and there are changes in the optical array, *disturbance of structure,* that are caused by movement and changes in the quality of light. What is problematic about both types of change is that they are not easily described by traditional mechanics or geometry [Gibson, 309]. I will explain how Gibson describes both types of change and show how Stiny's concept of a shape rule applies in each case. This will set the stage for describing how to create a collaborative digital environment for specifying, modifying, and navigating through digital content as shapes. Such an environment would allow for specific channels of communication to be more organically developed within them, just as they do in the physical environment, but the system would also allow for unpredictable and unclassifiable change to occur in contexts as needed.

Ecological Events

An ecological event is any change to the layout of substances and surfaces in the environment as opposed to in the structure of the optical array. When you observe the surfaces of my body move through the environment you are observing an ecological event, but me observing the door knob get bigger because I am getting closer to it is not an ecological event (that is a change in the optical array and will be discussed later) Ecological events are changes that are perceivable and thus only apply to a range of sizes, *"Ecological events, as distinguished from microphysical and astronomical events, occur at the level of substances and surfaces that separate them from the medium"* (93). In the same way that both Gibson and Stiny tried to avoid atomistic reductionism in the terms they selected for describing perception, Gibson is careful to point out that many types of change to the environment cannot be decomposed into simple mechanics and must be understood as changes in their own right. *"The laws of motion for bodies in space as formulated by Isaac Newton apply only to idealized detached objects"*. (94) So while some ecological events can be described this way, for example it is easier to describe some mass of substance in the form of a sphere moving through the medium as a ball, other types of change can not be so easily described by mechanics:

> At this level of analysis, the deformations and disruptions of a surface are not reduced to the motions of elementary particles of matter, either. Stretching-relaxing, for example, is an event in its own right, not a set of events; it is not reduced to a set of interrelated displacements of the elements of a surface. (96)

Two other types of change that Gibson addresses besides "change in the layout of surfaces" are "change in the color and texture of surfaces" and "change in the existence of surfaces". (Color and texture is beyond the scope of this thesis, although Stiny does have a theory that he calls *weights* which formalizes how color and texture can be assigned to shapes (Stiny, 219)) Change to the existence of surfaces is also not the type of problem that traditional mechanics or geometry can easily describe.

Stiny's goal with shape rules is very similar to Gibson's understanding of ecological events because he has created a way to represent and enact change that treats euclidian transformations, the introduction and elimination of structure, and stretching or bending in the same formal system. And although Stiny is concerned with shape rules for the purposes of architectural design and the composition of art, they can also be used to represent Gibson's ecological events in ways that text descriptions cannot. Here is a shape rule in the context of design:

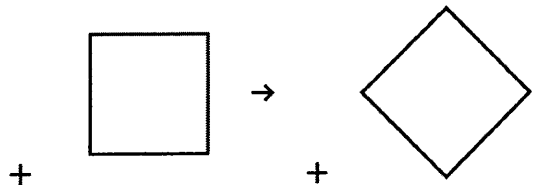


Fig 2.3.1 [F-MW]

For shape rules to be interpreted as rules and not just as a collection of shapes, two shapes must be seen as symbols and not as shapes, the arrow and the 2 loci. So, in the composition above I would say there are 15 maximal parts (for a formal definition of maximal parts see Stiny's text, but for the purposes of this text I am simply referring to discrete lines I can count) but as a linguistic convention I am interpreting some of the lines as an arrow and some of the lines as one of 2 loci. What is left after I have chosen to see some of the lines as symbols I would describe as objects, that is two squares both composed of four maximal lines each.

(As a quick aside, it is important that I make this distinction between seeing the loci and the arrows as *symbols* and the squares as *objects*. Seeing "squares" that I know to be composed of maximal lines is the same as describing some detached substance in the environment as a "ball". In both cases I am conscious that I am superimposing the idea of an object over what Gibson calls substance/surface and Stiny calls shapes. I do it because it is useful in the context of acting on whatever that stuff is. However, seeing symbols is me submitting to an established way of seeing as defined in a channel of communication; me and every one else who has studied shape grammars has agreed that those shapes *mean* arrow and loci.)

The arrow which divides the rule into a left space and a right space, indicates a *temporal* relation between a single space. Two loci are required so that the position of the shapes on the left corresponds in a known way to the position on the right. So, to enact the shape rule above in the context of drawing I would have to find an existing composition in which the left side could be *embedded,* that is fully traced out in an existing composition, delete that portion, and then replace it with the description on the right (229-230). If I take the right square and loci and superimpose them on the left square and loci it becomes obvious that the rule I drew can be described as "rotating a square 45 degrees around its center".

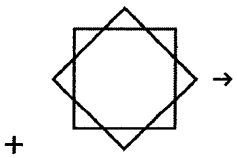


Fig 2.3.2 [F-MW]

But in the same way that picking out discrete objects is only useful in specific contexts, labeling the types of change is also just a convention. Some types of change that shape grammars could represent do not have such obvious English translations. Because a shape rule is enacted by first deleting the left side and then placing the right side, we would not say that whatever is on the left *became* whatever is on the right. This allows shape grammars to be easily used to represent all sorts of ecological events that other computational systems cannot. "1 + 1 = 2" is useful for talking about marbles but not balls of clay.

Of course shape rules are not useful, nor do they claim to be, in describing *why* change occurs. Is some stuff of the universe deleted at some instantaneous present and then replaced in some future with completely different stuff? Who knows, but what is important is that shape rules provide a formal and

useful way for *communicating* what is perceivable about change. Consider this compound shape rule below:

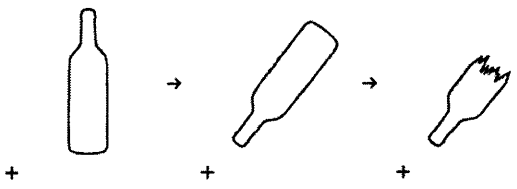


Fig 2.3.3 [F-MW]

This rule is not incredibly useful for the context of design because it is far to specific in its forms. However, as a means of representing change it is incredibly vivid and informative. Stiny would describe the rule in the most general scheme possible:

x -> t(x) -> part(x) + y

t(x) being a Euclidian transformation, part(x) being the deletion of some part of x and the preservation of another and y being a totally new addition. But a person who is not looking to see the formal description of the rule, but instead sees objects would say,

a bottle -> is rotated -> is broken

And when I look at the above shape rule/composition I also imagine a context, that someone in a bar grabbed a bottle and threatened someone else with that bottle by smashing it on the surface of the bar. I must admit I do not relate this context to the above composition because I have ever perceived such an incident in reality, but because I remember seeing images representing such an event in Hollywood movies. In this case I see information that is suggestive about the *affordances* of surfaces and substances that I have encountered before in my environment. Seeing shape rules vividly reminds us the type of change we have perceived in the environment in a way that symbols do not. But consider what we would say this shape rule suggests:

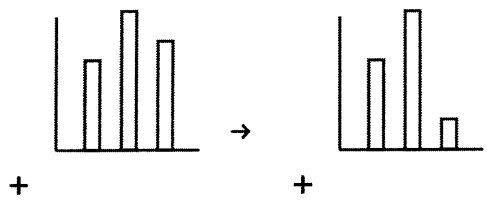


Fig 2.3.4 [F-MW]

In this case I would say I see a graph and it appears that the third bar appears to have gotten smaller. Here I am using these same maximal parts to show a change that is interpreted as a symbol within a defined communication channel just as easily as I am one that suggest a first hand ecological event. Because shape rules act only on shapes according to the measure of their content, and not according to their fixed meaning, they are useful mechanisms to describe unexpected change and the types of change that are predefined in set channels of communication. This is another reason why the goals of the Semantic Web must be reconsidered, by fixing the ways in which descriptions can change, they are presupposing that reality will change according to the types of change allowed by their predefined descriptions. When shapes are not treated as representations of data but as data types themselves, observing them change in form is observing both the computer description and the appearance change in meaningful ways. We do not need two webs.

Because shape rules can be used to create structure, delete structure, and change descriptions in unexpected ways they are a useful way to represent what Gibson describes as ecological events.

Disturbance of structure

Shape rules are useful for describing changes in what Gibson calls the optical array for the same reasons that they were useful for describing ecological events, visual structure is created and deleted as moving about the environment causes surfaces to come in and out of perception. But what is especially important about relating shape rules to the optical array is that Gibson is very clear that there is *more* information in perceiving how the array *changes* than there is in perceiving what he calls, *"an arrested optic array at a fixed point of observation (308)"*. Gibson calls this fixed vision, the kind that is represented in still images and perspectival paintings "snapshot vision" and says it is the least informative of the three other types, *aperture vision,* scanning the fixed array, *ambient vision*, looking and rotating ones head, and *ambulatory vision*, looking while moving (1).

Ambulatory vision is the only way to get information about what Gibson calls the *invariant structure*

> It is obvious that a motionless observer can see the world from a single fixed point of observation and can thus notice the perspective of things. It is not so obvious but it is true that an observer who is moving about sees the world at *no* point of observation and thus, strictly speaking, *cannot* notice the perspective of things. The implications are radical. Seeing the world at a traveling point of observation, over a long enough time for a sufficiently extended set of paths, begins to be perceiving the world at *all* points of observation, as if one could be everywhere at once. To be everywhere at once with nothing hidden is to be all-seeing, like God. Each object is seen from all sides, and each place is seen as connected to its neighbor. The world is *not* viewed in perspective. The underlying structure has emerged from the changing perspective structure. (197)

Perspectival changes are easily represented through shape rules as the optic array can be thought of in terms of 2 dimensional shapes, but how that change occurs simultaneously in relation to a non change in the layout in the environment is much harder to represent. Architects have tried to find ways to represent invariant structure through axonometric drawings. Below I have indicated two shape rules that are meant to show the change in the optic array and non-change in the structure that occurs when a perceiver moves away from a cube:

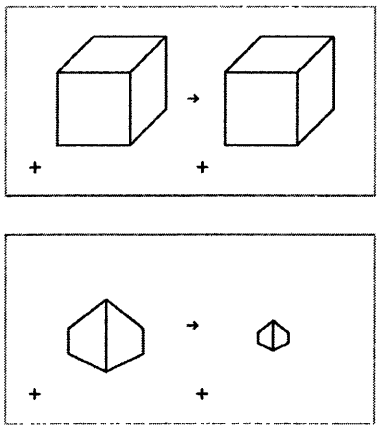


Fig 2.3.5 [F-MW]

In this case the cube is represented as an axonometric to indicate that I am attempting to indicate the form of the substance contained by the six surfaces independent of my own perspective. But this is a symbolic form of representation as no one has ever *seen* an axonometric type view of the environment.

A second problem that Gibson goes into in great detail is the problem of *occlusion,* that is the state of one surface being completely or partially hidden by another. If we use 2d shape grammars to represent the optic array then we lose this information, because in shape grammars there can be no overlapping of parts or else this would create unknown descriptions. In today's 3d rendering packages they already have developed the mathematics of projective geometry that can calculate the 2d representation of 3d geometric descriptions. However, it is only very recently in architecture school that 3d modeling and rendering packages have been able to output 2d geometric data as opposed to images in which the appearance is kept but the description is lost. The "make2d" command in Rhino allows users to generate 2 dimensional line drawings based on a 3d model and a defined viewpoint. Of course these files always need to be cleaned up because the resulting files tend to be rather screwy. Even still, the method is ubiquitous and is rapidly changing how students make 2 dimensional perspective drawings.

What the make2d command shows is how important it is to be able to describe the parts of the optical array and the parts of the environment with the same parts, that is shapes. Architects must consider both the optical array, *how buildings look,* and the invariant structure and composition, *how buildings are structured.* In architecture school a student might use a CAD package either to draw lines that are meant to represent a perspective, or the invariant structure through a conventional plan or section drawing. In either case the computer description of each is shapes and it is up to the user to coordinate the relevance between descriptions.

2.4. Summary

Gibson describes perception in terms of a medium, substance and surfaces. Gibson describes two distinct types of change, change to the layout of substances and surfaces in the environment, and changes due to the optical array caused by a perceiver moving or changing orientation. Stiny describes perception

as seeing shapes and develops the concept of a shape rule to represent any possible change. Shapes can be used to represent both substances and surfaces as they are laid out in the environment as well as the structure of the optical array. Shape rules also allow for special cases of change to be represented, that is parts joining, disappearing, dividing, or coming into being in both the environment itself and in the optical array.

However, a single composition of shapes or a single shape rule cannot fully capture the complexity of movement through the environment. A linguistic formalization claiming that two descriptions, one showing the perspective change and one showing the invariant change, placed in relation to one another, refer to be the same layout in the environment would need to be developed. In appendix 6.2 I give examples of my attempt to describe some of the types of change Gibson describes through shape representation.

3. Technical exploration - ContextCAD

3.1. Introduction

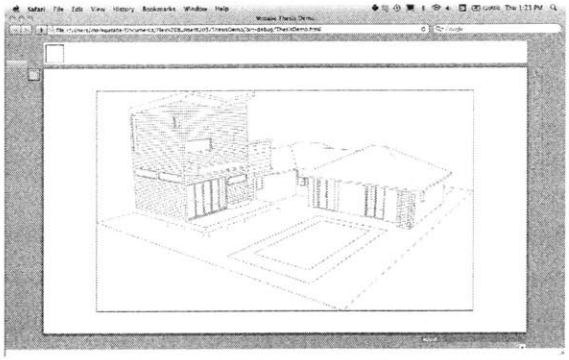


Fig 3.1.1 A single space in ContextCAD that contains 604 maximal lines of one color with no points, arcs, curves or subspaces. Given the way I have defined code, which I will explain in a minute, this space would not "compile".

The digital environment I have developed is an online drawing/coding program that allows users to define new spaces for drawing and the geometric parts that they might contain. As there are many aspects to this software, some that require new terms, and others that can be borrowed from existing methods in interface design, I have chosen only to represent a handful of topics. I will give an account of what I did, and describe what I did not do because it was either beyond my abilities or beyond the scope of this exercise.

I will refer to the demo as ContextCAD because one of the main arguments I am trying to make is that when users have the ability to define geometric parts and compositions within a single environment they do not need to be as concerned with fixing meaning. When we have the terms to describe what we are doing, our own observation of our actions in a context provides us with all the information we need. Users remember what they did, and an open system that allows them to draw shapes that are meaningful because of their appearance and ability to be compiled as code, and not because of their prescribed meaning, removes the need to think of code and the human readable layer as two separate layers. The computer ought to be concerned only with describing parts as they have been defined by users, and interfaces ought to only allow users to define parts that can be acted on meaningfully by the computer.

All graphic drawing programs that I know of, whether 2d or 3d, allow the user to both define geometry and navigate around an implied space to see the shapes from different points of view. The changes to the screen caused by "panning" or "zooming" could appear identical to the user moving shapes from one part of the space to another, and the user easily differentiates between the two because there is some indication of the type of tool he is using. A hand icon might indicate "panning" whereas an arrow might indicate some part is being moved. These conventions more or less simulate moving through an environment and for the most part my interface uses these same conventions. ContextCAD is a 2D environment and at this point only points and linear parts, not planes, can be defined within a space. The

focus of the investigation was not so much on making a complete drawing interface, nor on completely defining the programming language, but to make the first steps towards showing how it is possible to define a computing language based on shapes and pointing out many of the issues.

Another way to interpret this exercise is as an attempt to combine symbolic computation with shape grammars in a way that allows the principles of shape grammars to remain intact. However, a hard line shape grammarist would most likely see this exercise as a corruption of one of the basic principles, that symbols blind the viewer from just looking at what is really there. True, which is why I am trying to give the power of making symbols to the user. The power of the monitor is that it can change its dense matrix of pixels almost instantly. The composition can show anything, it is as if we have created 2d magic matter. Why are we not taking more advantage of this open ended-ness?

3.2. The parts

ContextCAD is an interface that allows a user to create new spaces, new geometric descriptions in those spaces, and spaces within spaces. Every space is parameterized from 0-1 on both axes and no shapes can be defined outside of the boundary either in part or in whole. The drawing interface is loosely modeled after Adobe Illustrator, and although the user does not have to refer to a shape rule to make a change to a shape, it must be the case that each instance of directly editing a shape description must be explainable in retrospect as the application of a shape rule. If a line is added the computer checks to see for possible fusing, and if a segment portion is subtracted the computer understands that a part might have been removed from within a maximal part.

Spaces are rendered with an outer glow, which also is the convention at the operating system level on my macintosh computer.

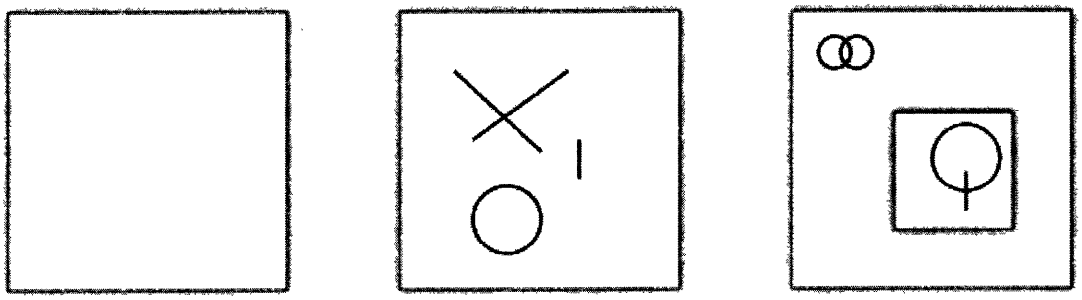


Fig 3.2.1 (left) an empty space (middle) a space with linear parts specified (right) a space with linear parts and one subspace containing linear parts

However, the concept of nesting spaces must be justified, as one goal of Stiny's theory of shape calculating is it to get away from hierarchical descriptions of reality. Also, as I am interpreting Stiny's shapes as the best way to indirectly represent Gibson's descriptions, I must also admit that Gibson does

not talk about nested *mediums*, even though he does talk about nested units. But my notion of nested spaces are not modeled on hierarchical descriptions of reality in the same way we think of a tree as being composed of a trunk, branch, and leaves. (Although my implementation of subspaces can easily be deployed to define a tree in that way). However, there is another aspect about how we experience trees that is useful for explaining my concept of a subspace. My notion of a subspace is based on the fact that a tree from far away appears as a *point* in the optical array, but a tree that one is climbing within and hanging from appears as an expansive *environment* or space. We cannot think of a tree then as a fixed object in the world, but instead we can say that "tree as point" points towards "tree as space". Moving towards the tree *as point* to experience it *as environment* can be called *dereferencing* the pointer. This type of subspace definition is very different than describing compositions as hierarchical.

All spaces have the capacity to have the same geometric parts defined within them. There is no limit to the levels of recursion. However, the shapes defined in one space do not relate to the shapes defined in another space, that is to say even if they are overlapping they do not fuse according to the rules of shape calculations.

3.3. The representation of subspaces

All spaces, regardless of their nesting status within other spaces all have the same 0-1 parameterization. Later we will see why this is important for treating shapes as code. Because the parameterization of all spaces is the same, subspaces are smaller then the superspaces they are defined within by virtue of the attributes that are assigned to them, such as their width and height in relation to the superspace. Shapes can not be assigned attributes beyond the specification of their layout and color because that would be as Stiny says, "treating them as points with attributes", which is exactly how subspaces are being treated. Subspaces can be rendered in four distinct ways:

1 As a space - a visible outer glow surrounds the specified geometry

2 As a represented point - some other set of shapes is represented in place of the specified shapes

3 As the *symbol* of an icon - to indicate *nothing more than* the subspace relation itself

4 As an object - the specified geometry is represented but not the indication of it being a space

The first type is quite easy to understand and an example was shown in the previous section.

The second type is similar to how icons on the desktop are currently used to represent either files, folders, or programs. In the case of ContextCAD I can think of several different motivations for selecting a geometric description to stand in for the one specified.

-When appearances don't indicate the function of code: If a subspace contains a complex layout of other subspaces and geometry that allows it to be compiled as code, than a more simple set of shapes that give me an idea of what the code does would be more useful than showing the code itself. The example on the next page shows a drawing that can be compiled as code on the right (whose function will be explained in the code section), and what I would choose to represent the code with on the left:

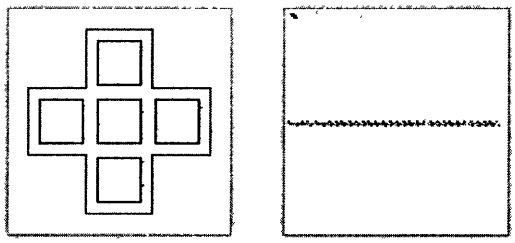


Fig 3.3.1 The left side set of shapes *stands in* for the right side set of shapes

-When significant differences are not *visually* significant and two things need to be differentiated.
-When I want the appearance of a space to not change even if its specified shapes do.
-When I want a more complex appearance to stand in for shapes that were selected for their minimal amount of information. This would include representing a single point in a space as a more complex glyph looking shape.

The third type is most important in establishing an understanding of the *semantics* of ContextCAD. The parts of the operating system are icons and unique spaces that are rendered with a drop shadow or other effect. And yet there is no icon for icon. I propose this icon as the icon for icon, that is for a subspace that is being represented by something other than its own specified shapes:

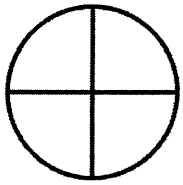


Fig 3.3.2 My proposal for the icon for 'icon'

The icon icon is important because graphic things *ought* to decompose into other graphic things. Wouldn't it be amazing if I could press a key and decompose the underlying structure of the graphically complex space I see to a more simple graphic composition:

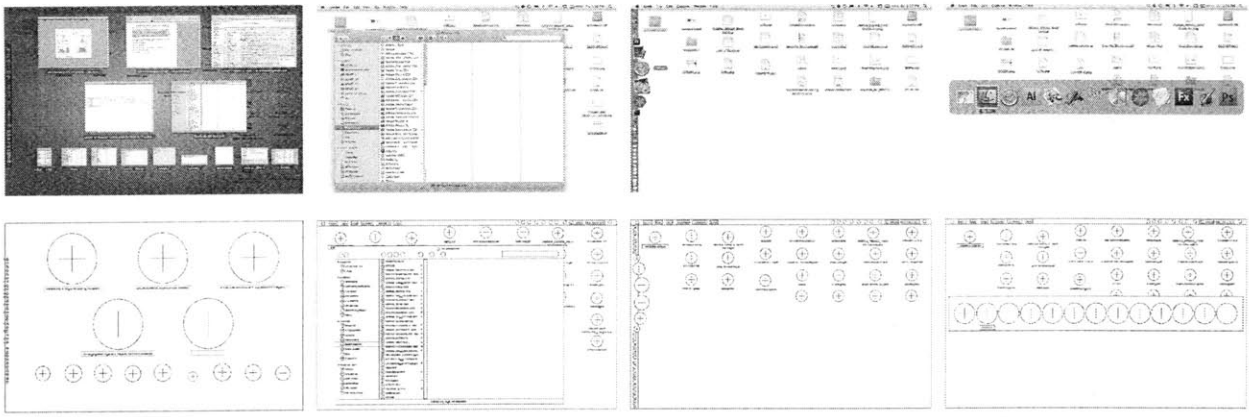


Fig 3.3.3 [F-MW] The shapes represented at the OS level are completely symbolic. Icons can be given X & Y coordinates against a background space which is at least *point like* behavior. (Screenshots taken from the Apple OSX operating system)

but instead when I select "view source" while surfing the web I see :

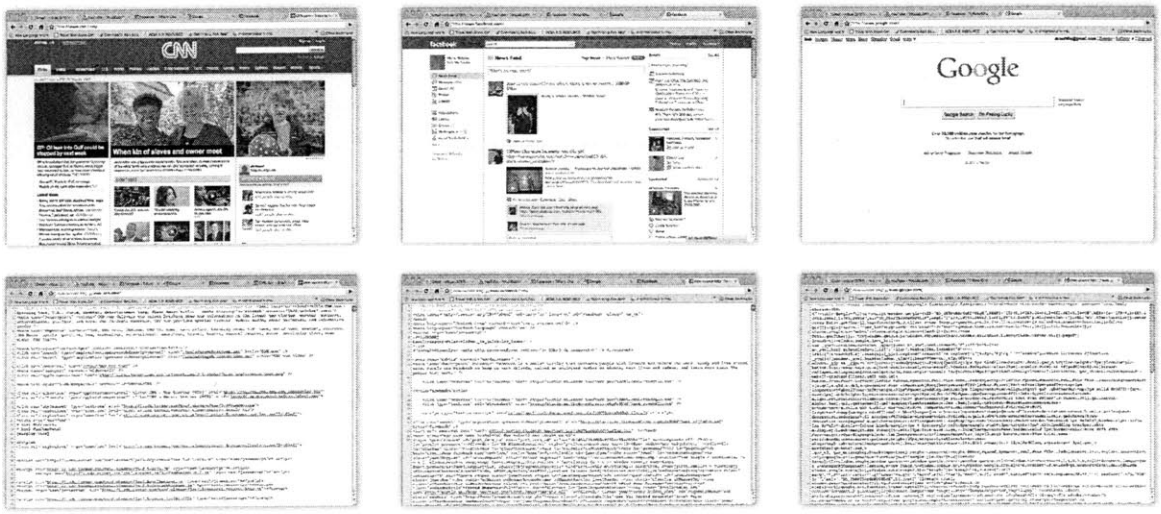


Fig 3.3.4 [F-2] Markup text is semantic only as markup text.

How does the source become the image? It is unknowable!

The fourth type allows maximal parts to seem as if they are part of a larger composition, but to be measured in relation to their own local space. The ability to switch between this mode, if desired, and the symbol of an icon is crucial when the shapes are considered data and not just representations of data.

3.4. Shape changes vs changes in the attributes of a subspace

As was explained, subspaces are all parametrically identical and yet can appear larger or smaller than other subspaces because subspaces can be assigned the *attributes* of width, height, x and y, as measured according to a superspace. This is the minimum number of attributes needed to render a subspace, although in ContextCAD they also have the attribute of *color* and *border-weight* which effects how their edge, when rendered as a space, is displayed. But I could have also given them many many more attributes such as rotation, bloated-ness, skewness, or any other value for which I am mapping to some effect that changes how the subspace is rendered. Changing the amount of rotation of a subspace is not the same kind of change as adding or subtracting a line; the rotation is just a number applied to the space that is represented graphically in a way that resembles what we think of as rotation.

The predefined ways in which a space itself can change I will call *the dimensionality of an object*, and it is very different from the dimensionality of the maximal parts of shapes. In ContextCAD spaces can be stretched or squished in the horizontal or vertical axis. This changes their *attribute* or *state* of "width" and "height" and not the parameterization or measure of their space. Every space is measured in reference to itself, one corner always being (0,0) and the other (1,1). Unlike with shape rules, the types of change that can be applied to spaces is predetermined and is saved as a "state". Shape descriptions do not have a "state", they simply appear as the specification of their layout in relation to a space so determines. The type of change below is the type of change that can occur to a shape description:

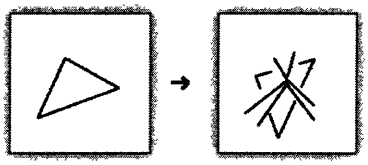


Fig 3.4.1 [F-MW] A change in the specification of shapes

It is not really classifiable. We might say "the triangle exploded" or "rays shot out from the center of the triangle" but the only thing conclusive we can say is "x -> part(t(x)) + y". However, in the drawing below we are building the type of change that we want to have possible for spaces into the system itself:

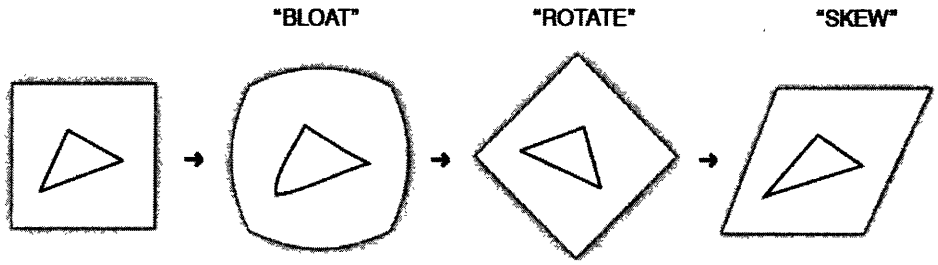


Fig 3.4.2 [F-MW] A change in the state of a subspace as an object

These sorts of effects are widely deployed in computers today, but they are most easily understandable at the operating system GUI level because we do not confuse this type of change with change in geometry as geometry is never specified in this root space. Consider the way that OSX shows the space of a menu being "minimized":

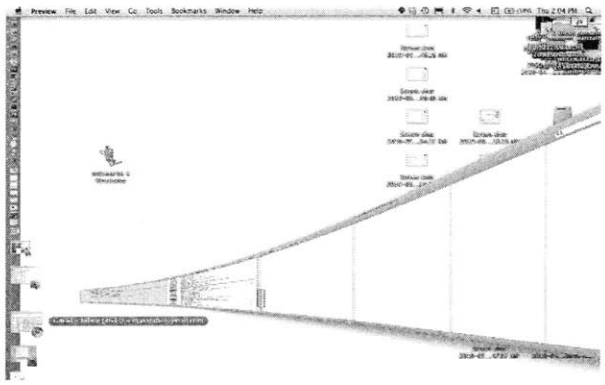


Fig 3.4.3 [F-SC]

Why is it that this effect is not nearly annoying as similar flashy animations on the internet? Because the layout and possible types of objects are so narrowly specified on the desktop such an animation is an effect for changing the state of a menu as a total space and not a change to some geometric description in the space. However, on the internet or even in specific software packages the types of parts that can be specified in the space are not so clearly defined, so it not clear what change the effect is actually indicating. But by clearly drawing a line between changes to shapes and changes to the representation of spaces more complex graphic transformations can be explored in various contexts for their potential usefulness.

3.5. How to store the parts

In ContextCAD I have made it possible to work with only points and linear parts. Linear parts are separated into three types: lines, arcs, and bezier curves. It is important that all of the maximal parts in a single space are saved in a strict linear order. The points are sorted in a list first by their x value and then their y value. The way I saved linear parts, however, is quite different from how it is normally done. In most CAD type programs an individual line might be its own object with attributes or history, but for the purposes of doing the most efficient shape calculations what matters for linear parts is most efficiently figuring out whether a linear part being added or subtracted overlaps with an existing part. For lines and arcs then, if one tiny segment or arc-chunk exists in a space then it is best to save it as a linear parameter of the largest possible part, lines that touch the boundary on both sides and arcs that are full circles. For arcs that are much larger than the space itself, I did not parameterize them according to the small angle that might actually be in the space, but instead checked for the boundary condition each time. The ordering scheme I used is explained below:

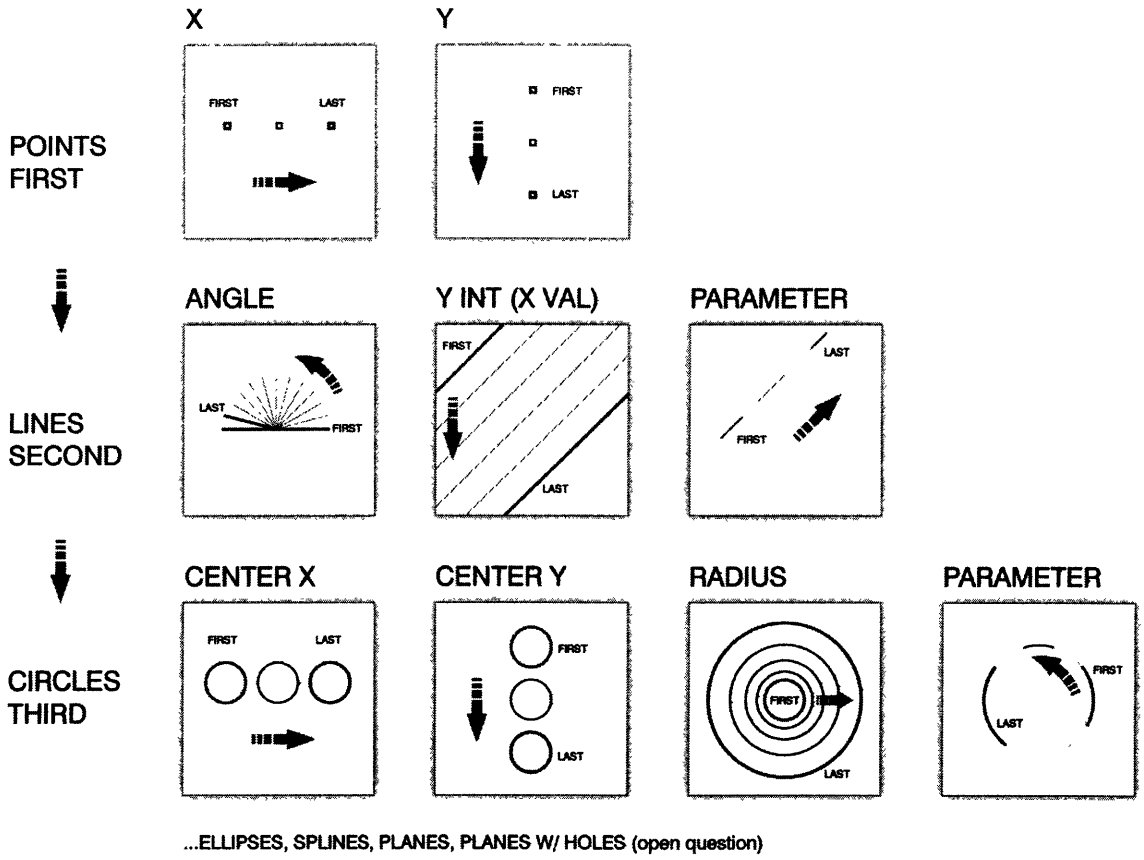


Fig 3.5.1 [F-MW] A scheme for saving all maximal parts in a strict linear order

I must point out many of the complexities that I avoided. I did not save parameterized values of extension or position as symbolic numeric representations, such as $\sqrt{2}$ or PI, and instead used floating point numbers. This of course meant having to check for slight variations when comparing two parameters or attributes used for ordering. A future investigation would be to try and connect this representation scheme to software such as Mathematica that does save numbers in a symbolic way.

Also for the curves I simply defined them by the four points that make up a cubic Bézier curve, and so they were not saved as part of a largest possible part. NURBS math would have been more useful but their mathematics are beyond the scope of this investigation. Finally, in this case I also saved color for each part and considered it as a "non spatial dimension". Two lines of different color do not fuse. Stiny would handle color by considering it as a weight and would treat it as value assigned to a shape. This would increase the complexity of adding or subtracting parts because the value of weights would also need to be added or subtracted independently of the adding or subtracting of lines.

A user does not need to be concerned with the code potential of a drawing he or she is specifying through the actual clicking and mouse moving that adds and subtract individual lines. Although a user is unable to specify data without it adhering this scheme. So, even if you were to draw this house without an intention of it being acted on as code, it could still be utilized as code. Of course the conventions and concepts I had to develop to make code work in this graphic only system make it so that code looks

nothing at all like this drawing. The two are defined by the same parts but that does not mean a user will confuse the different intentions behind the resultant forms.

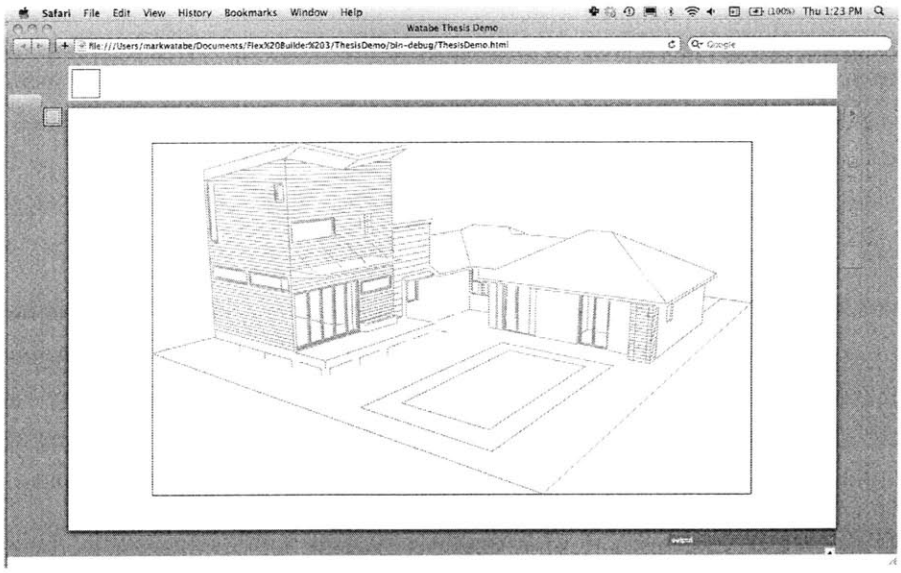


Fig 3.5.2 [F-MW] This drawing adheres to the maximal part principle and each part is saved according to the linearization scheme

4. How code works

4.1. Introduction

The basic premise of code goes like this:

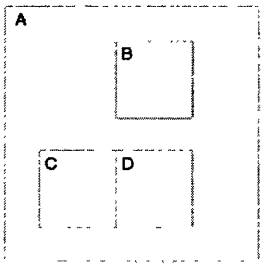


Fig 4.1.1 [F-MW] Automating the action of a shape rule in ContextCAD requires four spaces that be designated as the space of action (A), the space being acted on (B), the space specifying the shapes to be embedded on B (C) and the space specifying the shapes to be added to B (D).

Within space A the computer checks to see if the shapes of subspace C are embeddable in the shapes of subspace B. If not the computer evaluates to FALSE, and if so the computer evaluates to TRUE. In the case of TRUE the shapes of C are subtracted from the space of B and the shapes of space D are added to the space of B.

Assuming the following spaces correspond to the labeled space above, the drawing on the left would evaluate to TRUE and the drawing on the right would evaluate to FALSE:

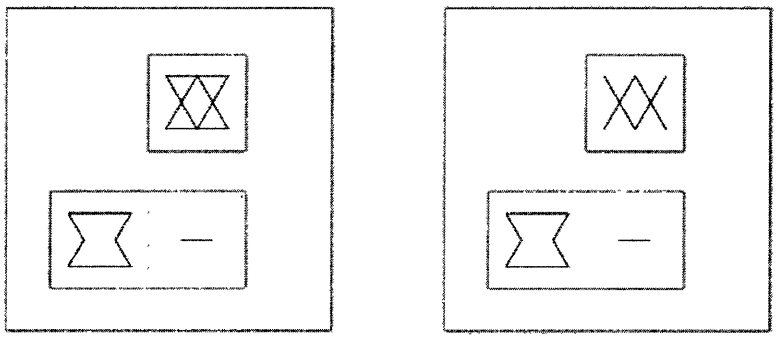


Fig 4.1.2 [F-MW] if seen as code the composition on the left evaluates to TRUE and the composition on the right evaluates to FALSE.

But how do these subspaces get labeled? In ContextCAD nothing has a name. Names tend to suggest unintended relationships between parts, or indicate specific purposes when there are many possibilities. The computer picks out shape rules in a superspace by looking at the layout of a circuit layer (that is

linear parts of a color designated for this purpose) in relation to the TRUE/FALSE condition of each operation.

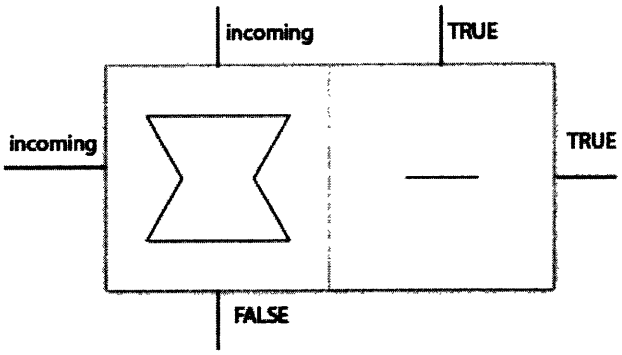


Fig 4.1.3 [F-MW] the flow of operations in the space is determined by the layout of the circuit.

After telling the computer to see a space as code, the computer picks the space for the left side of the first rule by seeing if any lines defined in the super space extend from its boundary to the top or left boundary of a subspace. This is an important distinction, this state of layout does not signify which space is defined as the first rule, the computer establishes where to start based on looking at the layout. Ignoring for a moment that we have not told the computer how to find which space to act on, let us assume it is some other known space, what the diagram above shows is how the computer will look for the next spaces to be interpreted as rules. If the left side can be embedded the rule is enacted and the computer looks for circuit lines extending from the top or right side of the right space to the top or left of another left side. If the left side can not be embedded then the computer looks for a line extending from the bottom of the left side to the top or left of the next left side.
To limit the clutter of lines I did not make the mechanism for finding the space to be acted on, (B), also a part of the flow of circuit lines. Instead I had to introduce a way to interpret some subspaces symbolically as *indirect references* to other subspaces. Also, to make code actually do more than what is literally specified I had to come up with a way to interpret spaces symbolically as *direct references* to some part of the shapes specified in other spaces.

4.2. Indirect & direct reference

The signifier is the pointing finger, the word, the sound-image. [ChangingMinds.org, 2010]

Gibson's concept of two types of information also suggests two types of reference, which I am going to call direct and indirect reference. The pointing finger is direct reference, one looks out at the environment and points towards features that can be picked out of the optical array. Yes it is true that the hand points towards *something*, perhaps the most red surface that can be seen, but the hand does not refer to some other space where a signifier and a signified are placed in relation to one another. The hand

simply points at what is directly seen. Indirect reference is when one refers to something like a map or dictionary. Both a map and a dictionary require confidence in the original author responsible for their composition. These two types of reference are formally defined in the way I have set up the coding language in ContextCAD. Indirect reference allows for the content of entire spaces to be referred to by virtue of their location in a superspace, and direct reference allows shapes themselves to be acted on based on their arrangement in a composition. In either case the computer must be programmed to "see" a space that is meant to be a direct or indirect signifier as a symbol that represents an action and not as a specification of shapes.

4.3. Indirect reference in code

When I press my finger down on the keyboard and a letter shows up in the white space on my monitor, such as I am doing now, I am not actually defining the geometry of a character in the space. My pressing the key specified a number in some list that is referenced against some chart, or *standard,* which eventually looks up the geometric data necessary to specify the appearance of a character on the screen. Below is a very small section of the Unicode Standard:

	000	001	002	003	004	005	006	007
0	NUL 0000	DLE 0010	SP 0020	0 0030	@ 0040	P 0050	` 0060	p 0070
1	SOH 0001	DC1 0011	! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071
2	STX 0002	DC2 0012	" 0022	2 0032	B 0042	R 0052	b 0062	r 0072
3	ETX 0003	DC3 0013	# 0023	3 0033	C 0043	S 0053	c 0063	s 0073
4	EOT 0004	DC4 0014	$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074
5	ENQ 0005	NAK 0015	% 0025	5 0035	E 0045	U 0055	e 0065	u 0075
6	ACK 0006	SYN 0016	& 0026	6 0036	F 0046	V 0056	f 0066	v 0076
7	BEL 0007	ETB 0017	' 0027	7 0037	G 0047	W 0057	g 0067	w 0077
8	BS 0008	CAN 0018	(0028	8 0038	H 0048	X 0058	h 0068	x 0078

Fig 4.3.1 [F-3] A small section of the Unicode standard.

What is interesting about the chart above is that the number information and the spatial information of the grid are redundant. If all of the 1,114,112 possible Unicode values were laid out in one giant 1024 x 1088 grid a person could refer to each little box containing some glyph by pointing to it (at the time that this thesis was written such a website in fact exists at http://ian-albert.com/misc/zoom-unicode.php). This is how reference works in ContextCAD. Subspaces and superspaces have the same parameterization, so any point of a subspace can refer to the corresponding position in the superspace. If a subspace contains a single point of some special "indirect reference color" (in this case I have picked magenta), then when the computer compiles the drawing as code it will check to see if that point, if considered in the same relative position of the superspace, overlaps with another subspace. Then the content of that space is retrieved. A simple drawing will help:

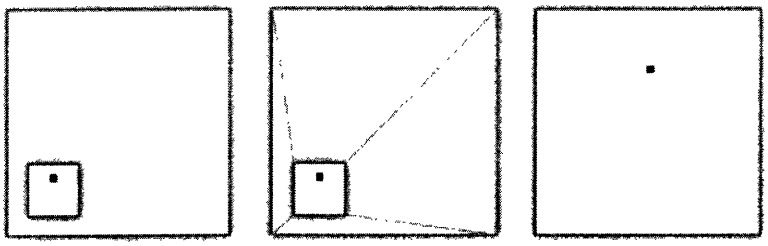


Fig 4.3.2 [F-MW]

The subspace containing a single point is scaled to fill the superspace.

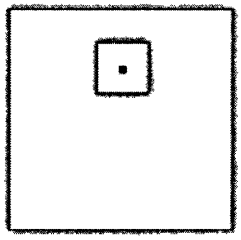


Fig 4.3.3 [F-MW]

But if another space was already in a position such that that space would overlap with the point,

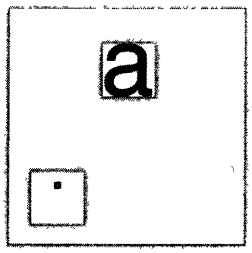


Fig 4.3.4 [F-MW]

Then we can say the bottom subspace *refers* to the top subspace.

This is an important aspect of ContextCAD, not just because it is how code automatically swaps out one description for another, but because it gives people the power to define their own graphic dictionaries. The entire purpose of the Unicode project is to make this process of reference invisible to the ordinary user. This is another example of how the people tasked with determining the standards for web communication go to great lengths to hide the computational process and produce graphics that are similar to familiar and global forms.

Of course the way I have set this up the reference could be to nothing, that is to say there could be no subspace that the referring point is contained within. In this case a set of no shapes is returned. I call this indirect reference because if we were to look only at the referring subspace, and not at the whole superspace, we could not for sure say to what the subspace refers. We would have to zoom out, scale the subspace, and look for ourselves if the point overlaps with another space. This is similar to when two people argue over the meaning of words. Either one person convinces the other to accept his definition, or they decide to *refer* to some other space where the letters specifying the "word" are placed in physical relation to the letters specifying the "definition". Then they can both point and look for themselves (this would of course be *direct reference).*

What I did not explore in this exercise and which needs to be further researched is how to make subspaces refer not just to the superspace they are in but to spaces on other machines. This would require a symbolic representation of the other machine to be placed in some specified relation to the subspace that is attempting to reference the space on that machine, and furthermore on that machine there would need to be a way to symbolically represent either any incoming space or a specific space on a specific machine.

4.4. Direct reference in code

When I work in drawing mode of ContextCAD I am constantly referring to different points and lines without describing to myself how it is I am referring to them. For example if I am playing with a field of 100s of points and I delete one somewhere in the middle I do not need to think to myself, "oh, that is the 30th one closest to the top and the 20th one furthest from the right". However, to get a computer to pick out specific points either these points must be assigned a name which is known by the computer, which is not possible in ContextCAD, or it must describe its position or attributes in some unique way.

And if we are to tell the computer that certain shape descriptions are meant to be seen as direct references, much like how a single purple point is to be seen as an indirect reference, then we need to come up with descriptions that are visibly easy to understand and general enough to be useful in code. Here are two examples I came up with that I am claiming refer to "point(s) closest to the bottom boundary" and "point (s) furthest from the left boundary":

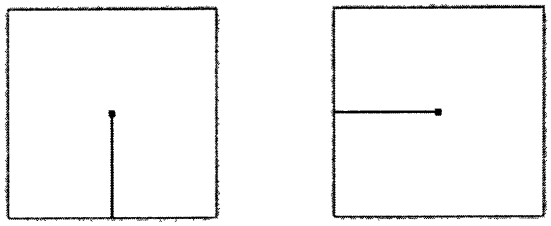
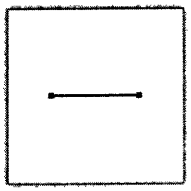


Fig 4.4.1 [F-MW] Two example subspaces meant to be seen by the computer as *direct references*.

For indirect symbols I picked the color purple to mean that a subspace should be read as an indirect reference. Here I am saying blue means "closest" and red means "furthest". If one of these spaces were to be specified as the left hand of a rule the computer would not try to embed what is actually there in some other space. Instead the computer would compare the description, such as "1 blue line, 1 point" to a list of predefined shape descriptions and act according to the actions specified in that list. Of course this list of operations is built into the system, just like how in programming the operators "+", "-" are built into the system. Of course I might not have picked the most essential relationships to signify, and the descriptions I have picked might not be the most computationally efficient. Nevertheless, in establishing how indirect and direct reference work I have a created a system in which I can define geometry that when acted on as code executes a known series of shape rules.

Direct rules can also be specified for the right hand side of a rule. For example a symbol such as this one:



Would not place those specified shapes, but depending on whether all points or all lines were embedded by the left hand rule, the computer would either place all possible maximal lines connecting all embedded points, or place all points that are the boundaries of all maximal lines.

At a zoomed out level code might look like this:

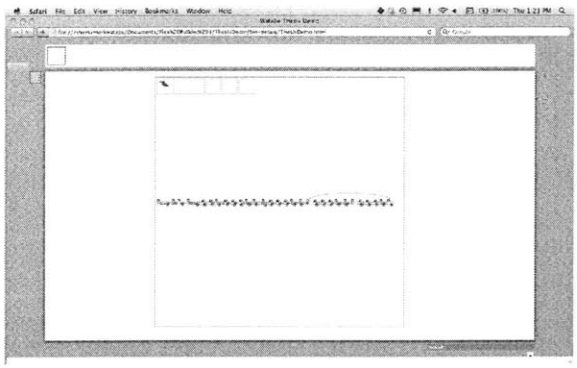


Fig 4.5.1 [F-MW]

And on closer inspection of the left side:

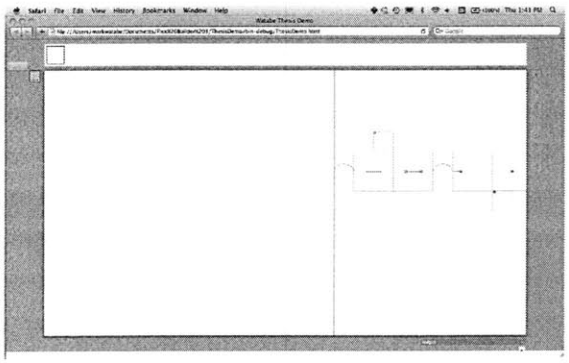


Fig 4.5.2 [F-MW]

Once a rule is discovered the computer then checks for spaces laid out adjacent to the rule, that is above or below, that indicate what space to act on. Or in the case of certain actions, such as *"embed the point in A that is closest to this point defined in B"* both A and B are laid out as indirect references. If the code is malformed in any way the computer simply stops. This is different then evaluating to FALSE, which is a part of the logic of the code.

4.6. Example code

The example I am going to show in this document is based on a set of shape grammar rules that George Stiny and Bill Mitchell specified in an essay called "The Palladian Grammar" [Stiny & Mitchell, 1978]. I did not pick this example after I designed how the code would work, but it was one that helped me design the code. I also experimented with trying to do simple force calculations on bridges based on shape relations alone and not numerically, but have not included them . Because the actual drawing

interface was not developed to a user friendly level I would first draw in Adobe Illustrator while imagining the calculation in my head, and then hard code it into the interface to see if it worked. Here are the original rules defined by Stiny and Mitchell:

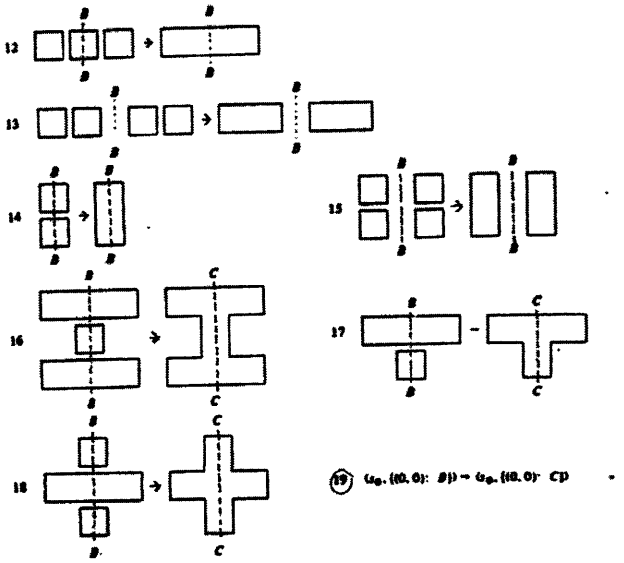


Fig 4.6.1 [F-4] This collection of rules suggested to me a more general algorithm for tracing the boundary of discrete squares.

It seemed to me that each of these rules was an instance of some more general operation that would also cause these sorts of changes:

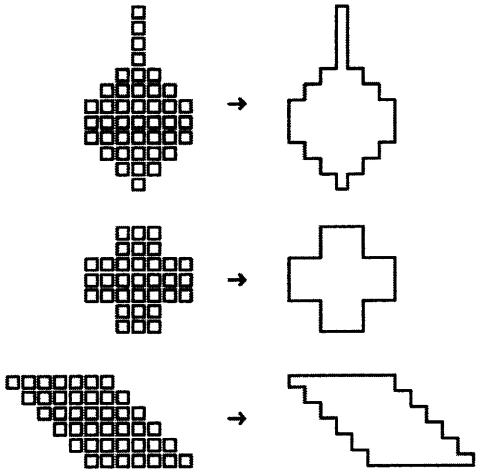


Fig 4.6.2 [F-MW]

Trying to figure out how this was possible with the way I had defined code was difficult. Setting up an automated shape rule that enacts rules on one shapes on then on another didn't seem to provide a

way to cause this type of composition to be developed based on simple moves. It became easier once I introduced this convention:

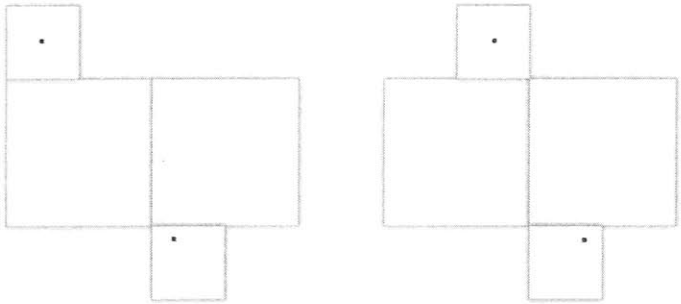


Fig 4.6.3 [F-MW]

In this case the space that refers to the space to be acted on also indicates whether the geometry to be embedded ought to be deleted or not. With shape rules as defined in shape grammars the left side is always deleted and replaced by the right side, but I found it necessary to make this a specifiable option. Also the space to the right bottom might indicate a different space than the top left one to put the geometry in. Of course if the left side does not necessarily delete what is there, and the right side does not necessarily act on the same space, what I have defined is really not a shape grammar at all.

My original hope was to act on lines as lines, but it proved much easier to decompose the entire drawing into points and do a series of rules specifying "embed point(s) furthest from" and "embed point (s) closest to" rules over and over. The long band sections contains the rules and the spaces at the top are used to isolate points and draw lines.

Fig 4.6.4 [F-MW] The long band included one loop that was true so long as the the point at the top most left corner taken from the boundary of the input geometry remained in the original space, otherwise it evaluated to false and the operation stopped.

What was interesting is this fact, although my code rebuilt the outline in the smallest parts, the final representation was composed of maximal lines. This is significant because it means that with code or with the interface I was guaranteed to have well defined data given that I was clever enough to figure out the visual code or willing to draw the shapes manually.

5. Conclusions

5.1. Summary and discussion of work

Unfortunately my attempts to make coding *easier* to visually understand have not been realized. We think of coding as giving commands, we give variables and objects names so that we can quickly refer to them without thinking about where they are located or about extraneous information about them. The visual coding I developed required code to mostly be about extracting a point from a collection, putting it in its own space, acting on the point in some way, and then placing it into another space.

What I do consider a worthwhile contribution was the investigation into explicitly defining 2 types of reference and striking a fine line between interacting with shapes and interacting with symbols on the computer. A shape grammar interface would not make sense in the context of how computers are currently utilized, it would be the end point of so many symbolic references. Shapes as shapes, if we consider them to be the best way to represent the substances and surfaces that compose direct reality must be seen as the underlying parts. Shapes can be utilized to represent both the environment as it appears and all of the symbolic forms humans have created to communicate with one another. It is not code, bits, atoms, or 1's and 0's that compose reality, visual reality at least is the surfaces at the boundary of the medium and the substances of the environment, and the best way to represent this reality is with shapes as defined by Stiny.

Human acts are computed in unknown ways to cause the monitor to change its composition, we need to stop talking about the internet as if it is simply a matter of privacy for sharing "information".

5.2. Future work

There are many aspects that I was not able to implement that might have made the process much easier and provided unexpected benefits to this method of coding versus coding with text and numbers.

For one, ContextCAD is meant to be a networked platform. I hypothesis that multiple people working on one large composition of shapes would be a very different experience from multiple specifying symbols in separate spaces. For one thing, it is quite easy to make a representation of other people and what they see in ContextCAD.

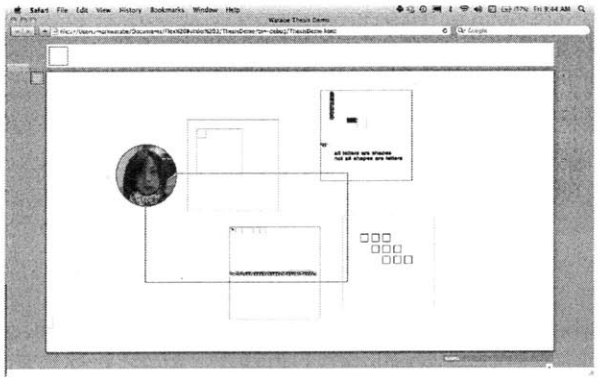


Fig 5.2.1 [F-MW] Representation of another person working with the same shape descriptions.

Consider that in online collaborative video games people in which people converse with one another they are usually looking at graphic compositions and not text, and this allows them to separate their conversation from their visual enabled actions in the virtual environment. Would a purely visible code allow people separate their visual and aural modes of perception and increase their ability to communicate about coding while actually coding? I think so.

The second thing I was not able to fully implement was the different modes of representation for subspace types. Being able to represent a space by some arbitrary geometry would have allowed me to more easily label the parts of my code. This would have really helped managing the movement of points in drawing the boundary of the collection of squares.

However this thought also seems to defeat the purpose of the whole endeavor. This relationship would have to be defined somewhere. But in what space?

The third I started but did not fully explore was working more with text forms. All letters are shapes but not all shapes are letters, so it seems if this system were developed enough it should be possible to create a text editing environment. But why text besides the fact that it is what we have always done?

The fourth exploration I started but was unable to implement in this period was showing how shapes as information could be understood as the information that passes from sensors coming into the space and to machines going out of the space. For a laser cutter it is obvious that the data sent is a shape (with some metadata of course) but for something like the temperature we think of it as a number and not as a point on a line. But if subspaces in ContextCAD could be connected to sensors which specify shapes just like humans or code does it would be possible to not only connect physical sensors and actions in the world but also to see these events occur over time without the need for an interface to show this "numeric" data.

And finally I must say that this entire project also comes from my previous work based on developing a time zoom interface. In ContextCAD it would be possible to define time as a very stupid person that just moves a point at some interval. Then other sensors or people that act in relation to those changes would appear to be defining shapes representing temporality, such as timelines. This would make the changes that "computer as timekeeper" causes appear on the same setting as changes that other people cause. This would bring us one step closer to perceiving the environment as described by Gibson:

> *Events are Primary Realities* In the first place, the flow of ecological events is distinct from the abstract passage of time assumed in physics. The stream of events is heterogenous and differentiated into parts, whereas the passage of time is supposed to be homogenous and linear. Isaac Newton asserted that "absolute, true, and mathematical time, of itself and from its own nature, flows equably without relation to anything external," But this is a convenient myth. It assumes that events occur "in" time and that time is empty unless "filled." This habitual way of thinking puts the cart before the horse. We should begin thinking of events as the primary realities and of time as an abstraction from them --a concept derived mainly from regular repeating events, such as the ticking of clocks. Events are perceived but time is not (Gibson, 1975). (100)

6. Appendices

6.1. References

ChangingMinds.org, (2010) "Signifier and Signified"
http://changingminds.org/explanations/critical_theory/concepts/signifier_signified.htm

Fensel ,Dieter ... [et al.]. (2003) *Spinning the semantic Web : bringing the World Wide Web to its full potential.* Cambridge, Mass., MIT Press.

Gibson, James J. (1986) *The Ecological Approach to Visual Perception.* New York, NY., Psychology Press

Stiny, George (2006) *Shape*. Cambridge, MA., The MIT Press

6.2. Image references

[F-1] Munroe, Randall. **Computer Problems,** xkcd
http://xkcd.com/722/

[F-2] Three screenshots showing the source code of three different websites on May 20, 2010
http://www.cnn.com/
http://www.facebook.com/
http://www.google.com/

[F-3] http://unicode.org/charts/

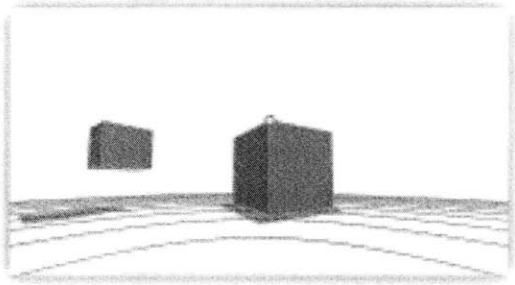
[F-4] **Stiny, G., & Mitchell, W. J.** (1978). "The Palladian Grammar". *Environment and Planning B: Planning and Design vol 5*, 5-18.

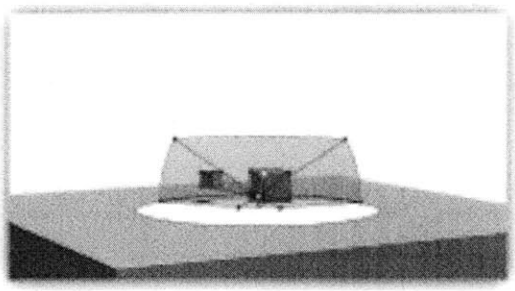
[F-MW] Image courtesy of Mark Watabe

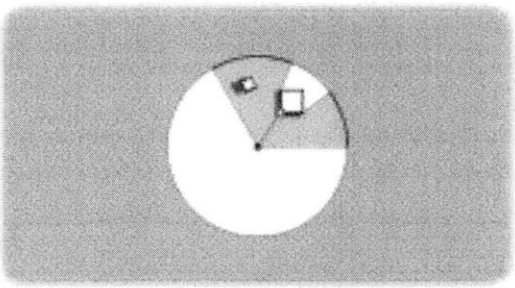
[F-SC] A screenshot of the operating system of my own computer

6.3. Supplementary diagrams

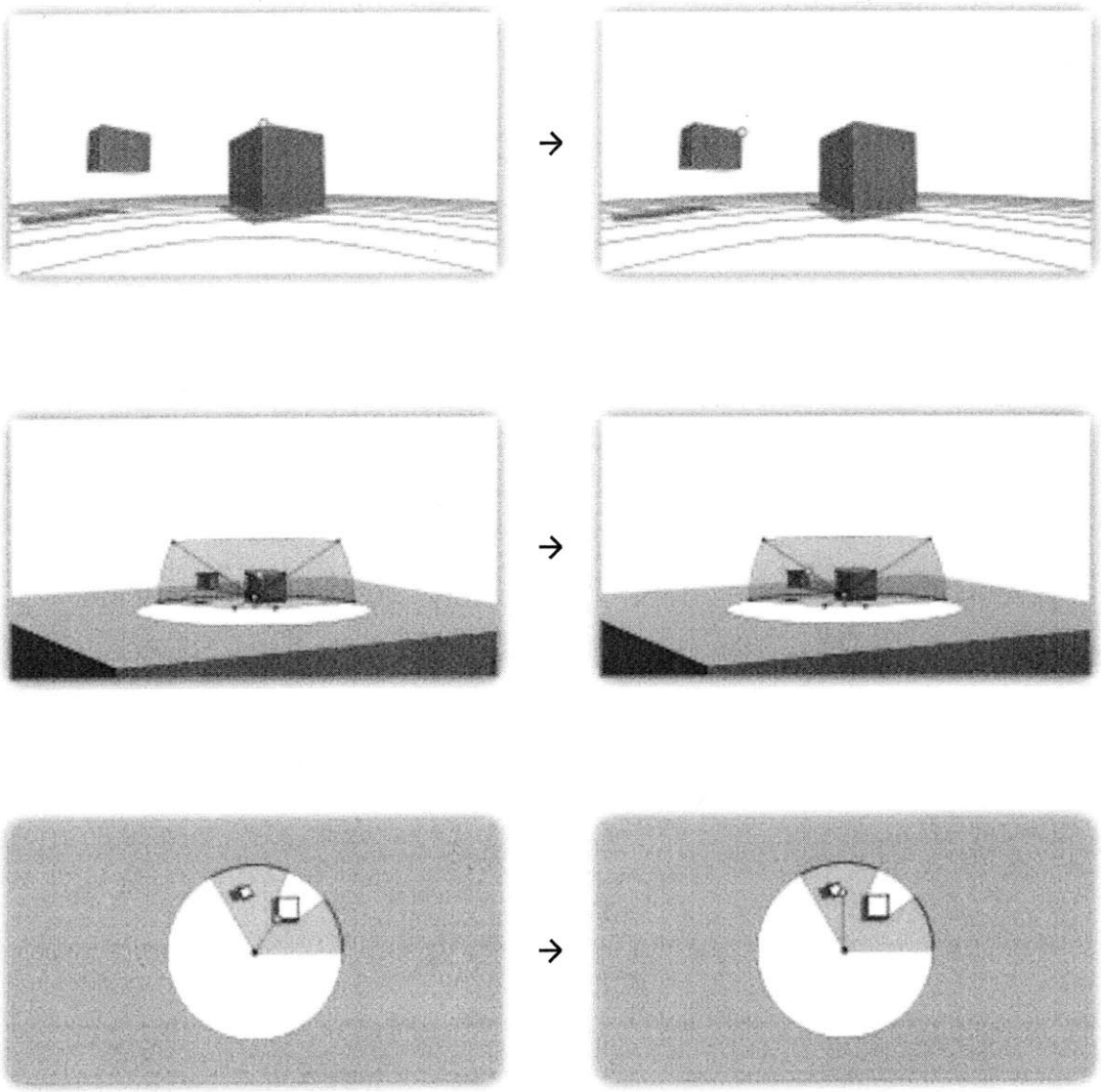
The following 8 pages show my attempt to represent Gibson's classification of 4 types of seeing, and 4 special cases of when surfaces go out of existence in the optical array but not in the environment.





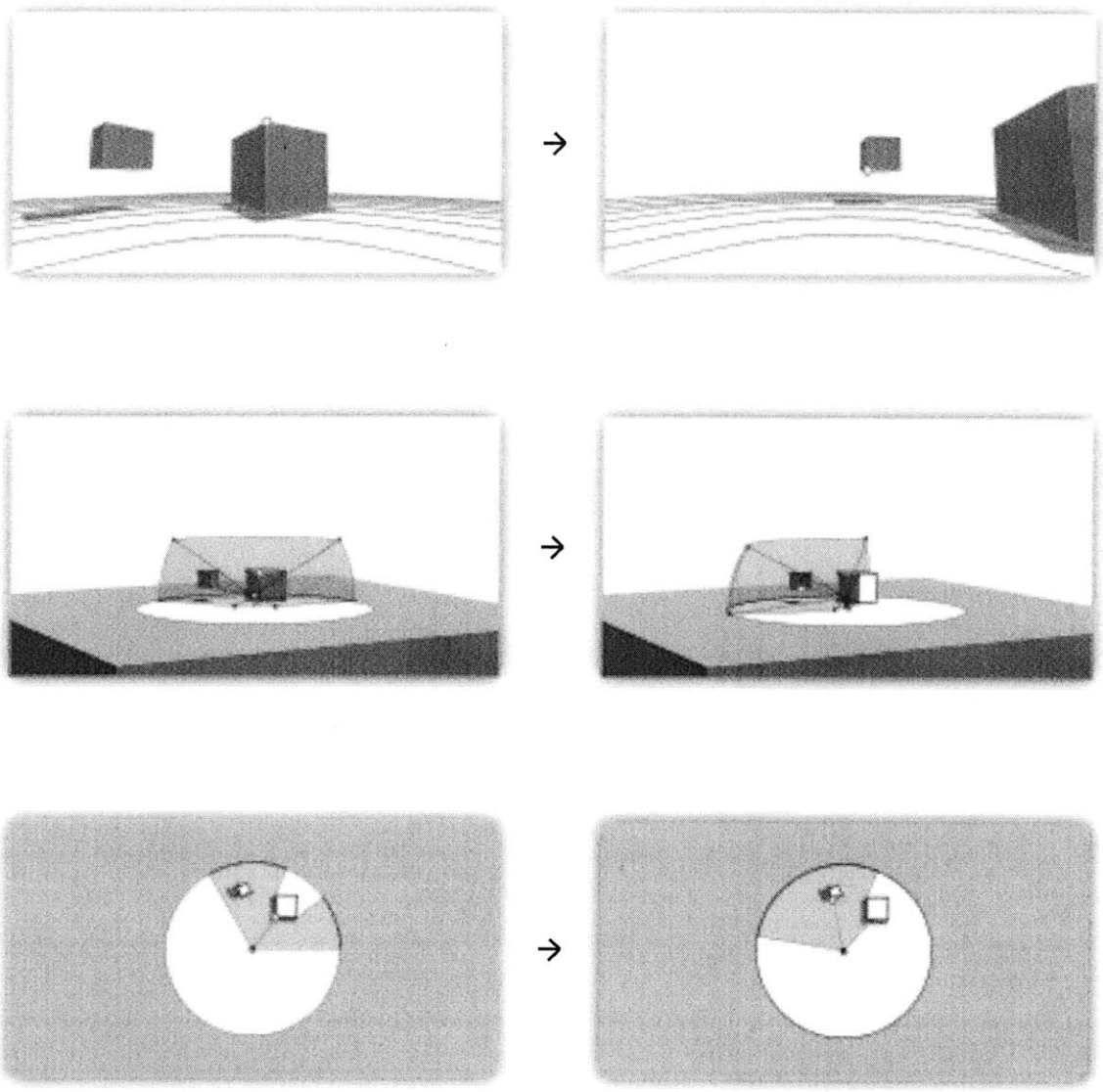


snapshot vision



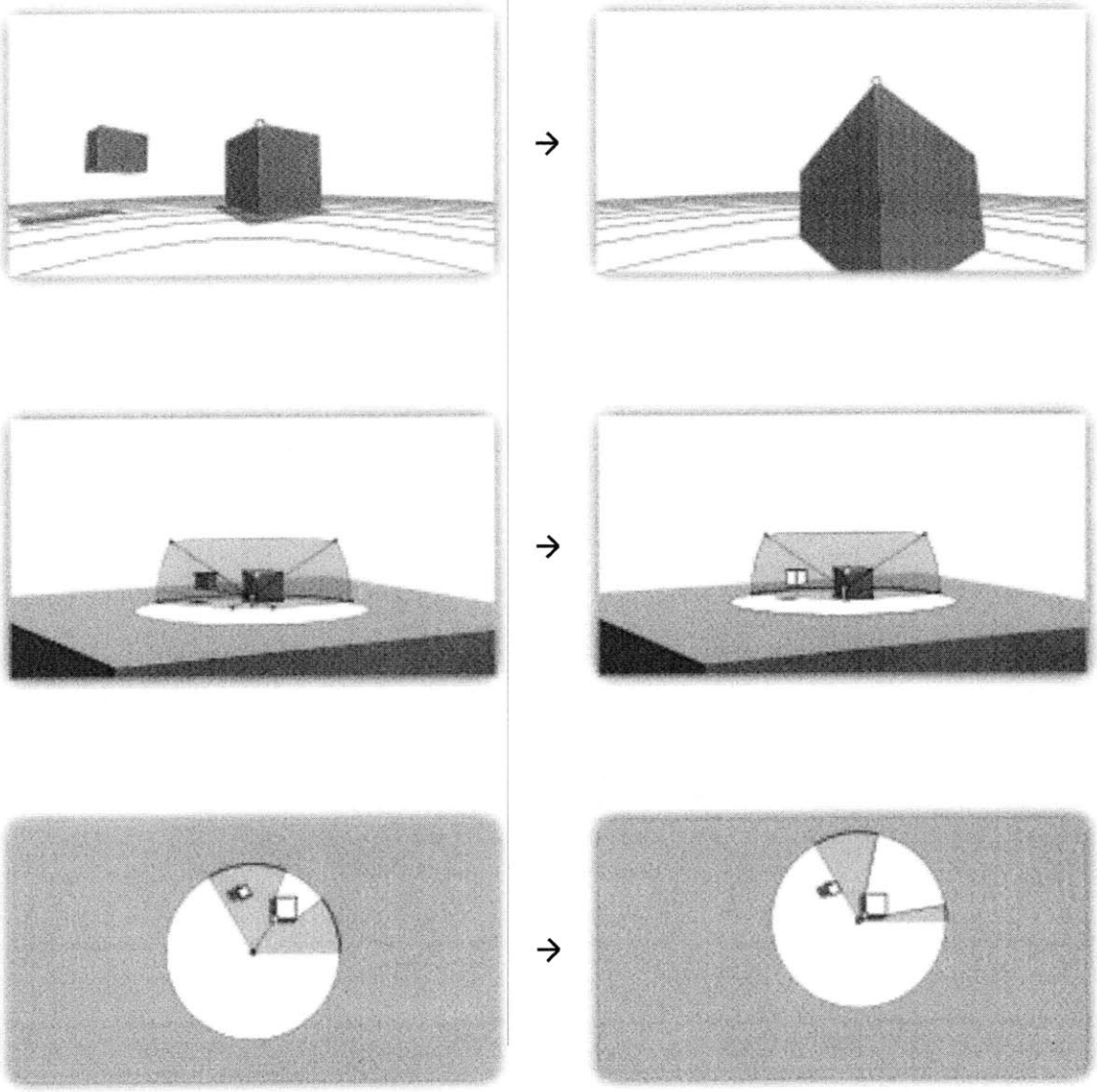
aperature vision

a different point became the point of focus.



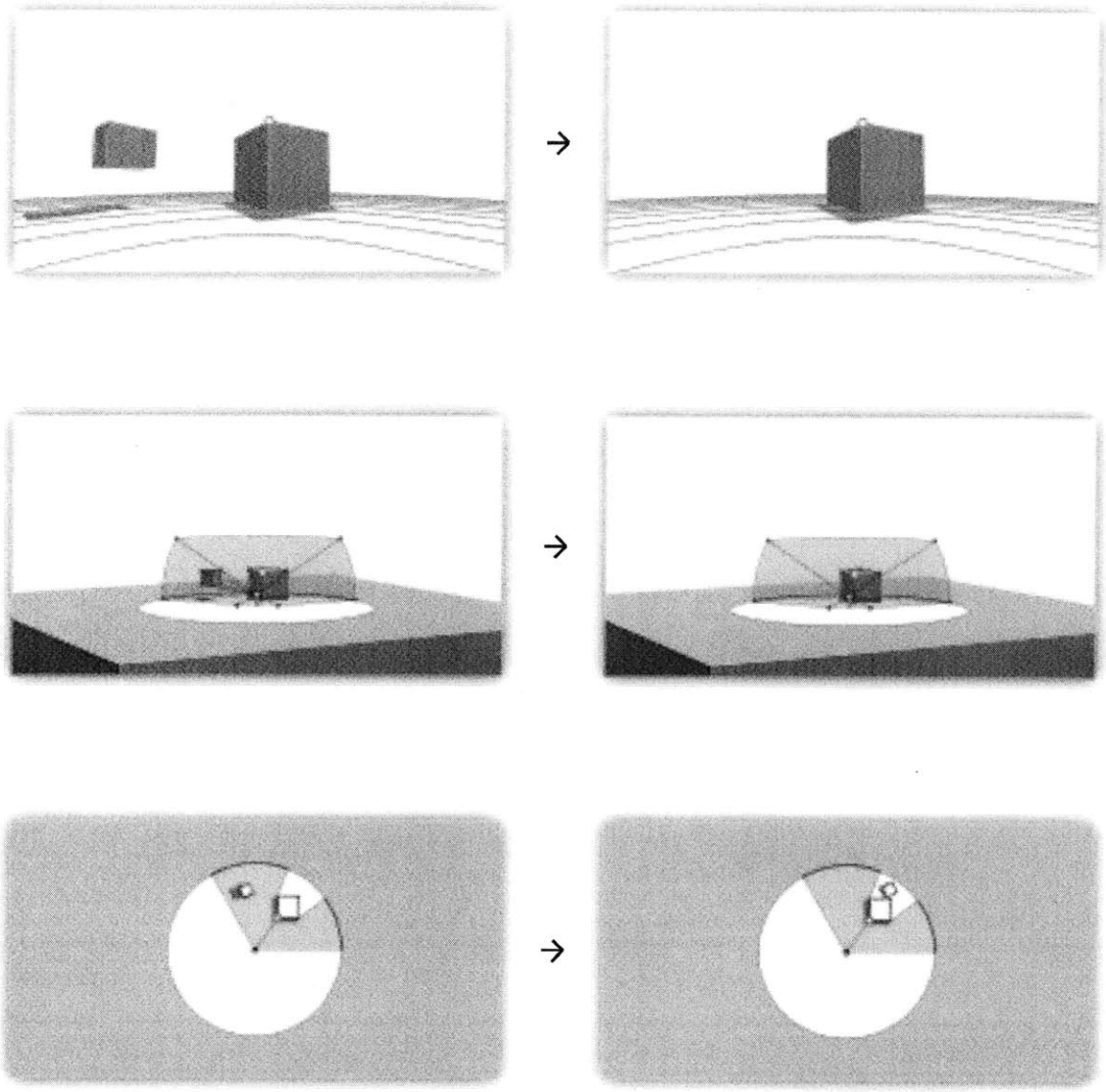
ambient vision

the head was rotated. a different point became the point of focus.



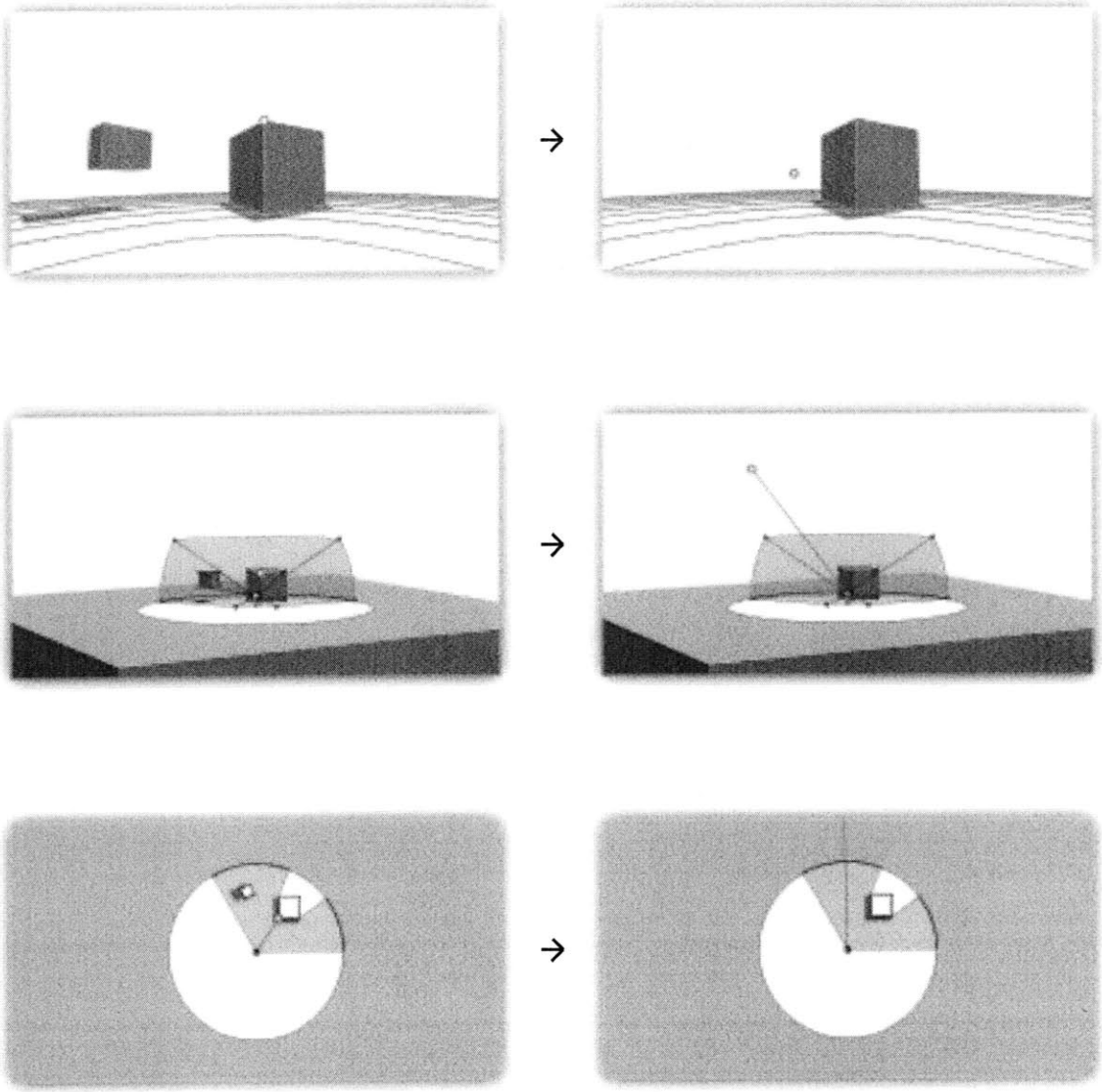
ambulatory vision

a self moved in the environment



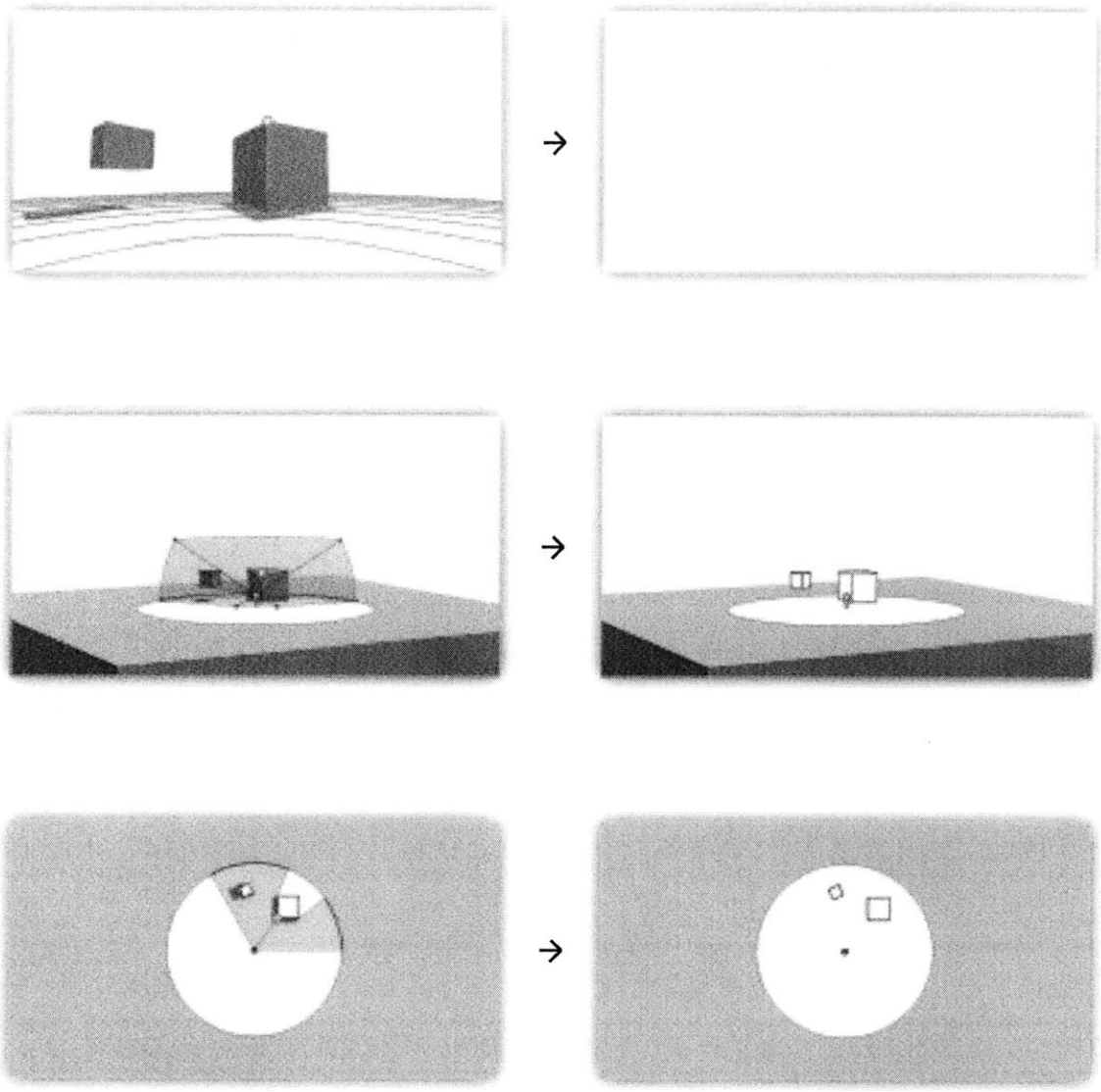
a surface disappeared

a surface went behind an occluding opaque surface



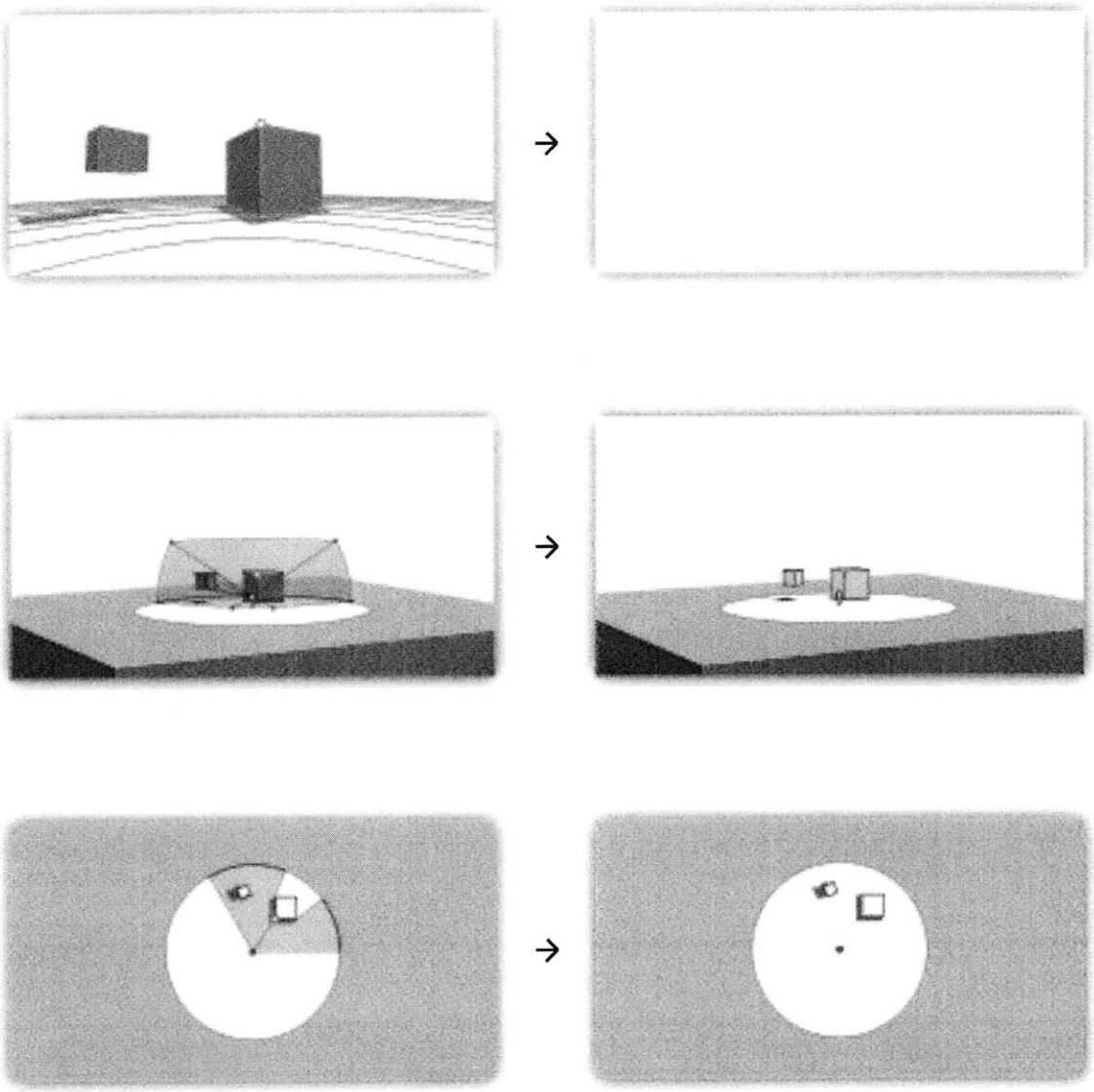
a surface disappeared

a surface minimized to a point by increasing its distance from a self.



a surface disappeared

the illumination in the environment was reduced to 0.



a surface disappeared

a self closed its eyes