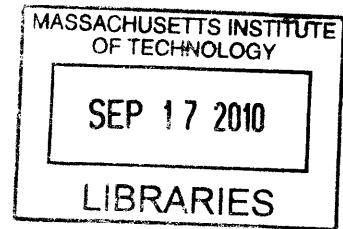


DURATION, DENSITY, AND EVOLUTIONARY FORM  
APPLICATION OF BIOLOGICAL PRINCIPLES TO ARCHITECTURAL SURFACE



JOHN ROTHENBERG

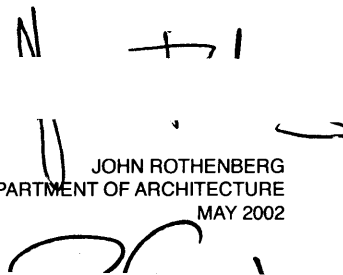
SUBMITTED TO THE DEPARTMENT OF ARCHITECTURE IN PARTIAL  
FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF BACHELOR  
OF SCIENCE IN ARCHITECTURAL DESIGN AT THE MASSACHUSETTS  
INSTITUTE OF TECHNOLOGY.

ARCHIVES

JUNE, 2002.

©2002, ALL RIGHTS RESERVED, MASSACHUSETTS INSTITUTE OF TECHNOLOGY

SIGNATURE OF AUTHOR

  
JOHN ROTHENBERG  
DEPARTMENT OF ARCHITECTURE  
MAY 2002

CERTIFIED BY

PETER TESTA  
ASSOCIATE PROFESSOR OF ARCHITECTURE  
THESIS ADVISOR



## DURATION, DENSITY, AND EVOLUTIONARY FORM

### APPLICATION OF BIOLOGICAL PRINCIPLES TO ARCHITECTURAL SURFACE

Changes in the way we look at the relationship between artificial and natural in architecture lead to new design possibilities that incorporate the ideas of organic and evolutionary form. When these biological models are coupled with computational power, architectural design can begin to address the dimension of time. By dealing with the concept of duration, architecture reaches levels of complexity that match the world in which it exists. This thesis will explore the possibilities of organic and evolutionary surface in architecture and will attempt to provide a design that responds to a new understanding of duration in architecture.

This thesis explores the relevance of a time-based model of architecture that draws on the concepts of organic and evolutionary form. It proposes a new understanding of architecture's relationship with time, and then uses this concept of duration as the foundation for experimental design.

The goal of such an exploration is to gain a deeper understanding of the physical consequences of a new theoretical approach to architecture. A secondary goal is to demonstrate the power and necessity of computation in collaboration with biological models of architecture.

The thesis includes research into nonlinearity as a challenge to the dominant understanding of the relationship between artificial and natural. From this research, the concept of duration is proposed as a new way of viewing the distinction between the natural and artificial. Duration then becomes a metaphor and guide in the design process of a studio assignment. In order to expand the possibilities of this concept, the thesis looks to ideas of adaptation and evolution as well as biological models of systems exhibiting these properties. Computation is used to model design sketches based on adaptive and genetic algorithms. Finally, a design experiment incorporates all of these ideas in an evolutionary architectural surface.

## CONTENTS

- I. PROJECT BACKGROUND: DURATION AS DISTINCTION
- II. FOUNDATION: DESIGN WORK INFORMED BY DURATION
- III. COMPUTATIONAL APPLETS: ADAPTIVE AND EVOLUTIONARY
- IV. ADAPTIVE SURFACE: MOVEMENT AND MEMORY
- V. CONCLUSIONS: ISSUES OF DENSITY AND INTENSITY
- VI. ADDITIONAL MATERIALS: SOURCE CODE



## I. PROJECT BACKGROUND

PROJECT BACKGROUND  
DURATION AS DISTINCTION



TODD HIDO  
UNTITLED #2621, 2000  
PITTSBURGH, PA  
CHROMOGENIC PRINT  
24 X 20", 38 X 30", 48 X 38"



## RETHINKING THE DISTINCTIONS BETWEEN MAN AND NATURE

Fundamental building and design principles are based on a certain understanding of the relationship between man and nature. If our understanding of this relationship were to break down, so too would much of our current architectural dogma. Furthermore, when this relationship is challenged, a new set of design possibilities emerges. Most clearly, a new understanding of the problem of man and nature in architecture questions the way we look at duration -- how time relates to the built world. Where theory breaks down, there is the possibility of design at the point of discontinuity. Old relationships are replaced by innovation.

The initial problem with the relationship between man and nature stems from the existence of this distinction. Architectural theory fails to account for the possibility that there is no distinction between man and nature - that both are components of the same system. Let's look to the current paradigm. Although there are myriad opinions on this division, we tend to set up two categories, and then place elements of our world into one of these categories. First we look to absolutes: the places least touched by civilization (the depths of the ocean, remote and inhospitable land), are categorized as part of the natural world, while the places most altered by the hand of man (cities, dams, bridges) are categorized as part of the artificial or built world. {Footnote: The word artificial is hardly ever used, but it is the literal opposite of the natural, and using it helps to clarify the underlying leanings of this system of categorization.} We follow this by adding to each category, guided by the idea that anything not touched by man is natural, and anything constructed by man (or his intellect) is artificial. Technology, or its presence, also becomes an indication of the artificial. Certainly cities and technology are built, but this system of categorization presupposes that they are artificial. A question naturally arises: is the built world necessarily artificial? So we actually have two distinctions: one between the built and the unbuilt, and one between the natural and artificial. The problem originates from our combination of these two distinctions and the resultant architectural dogma. When we group the unbuilt with the natural and the built with the artificial, we confuse two entirely different questions.

Contemporary discourse and scientific discovery have found that the quantitative differences between man and nature are less dramatic than we once thought. The Human Genome Project demonstrated that man is not more genetically complex than the rest of the animal kingdom. In fact, the genetic variation between humans and other animals is minimal, and humans do not possess the greatest number of genes. Even the distinction between living and inert matter seems to be blurring. The author and Columbia University lecturer Manuel De Landa discusses this in his essay "Nonorganic Life." As computational power increases through technology, we have been able to perceive the dynamics of more complicated nonlinear chemical relationships. Previously, it was thought



TODD HIDO  
UNTITLED #2840, 2001  
KENT, OH  
CHROMOGENIC PRINT  
24 X 20", 38 X 30", 48 X 38"

that all chemical reactions moved linearly from one state to another. In fact, certain systems have demonstrated the presence of inert matter that exists in an infinitely dynamic state -- one that doesn't require a constant source of energy. It seems that nonlinear relationships exist throughout much of the sciences. Progress in the built world has allowed us to explore these relationships.

Thus, one of the main causes of the paradigm shift that has allowed us to "see" matter as capable of self-organization is an advance in the technology that materially supports mathematics, and with it mathematical technology. Needless to say, this will not be an overnight replacement, and much of science (classical and quantum) will remain linear. But nonlinear science has begun to reveal new and startling facts about matter, in particular, that the behavior of entirely different material systems can exhibit "universal features." [De Landa, "Nonorganic" p134]

Much of these phenomena remained invisible until the end of the twentieth century. The difference between organic and inorganic matter is deteriorating. We are finding that traditionally inert matter exhibits certain properties of life. "In short, it seems that our bodies are inhabited as much by the phenomena of "nonorganic life" as by the more familiar phenomena of organic life." [De Landa, "Nonorganic" p133]. These discoveries have a drastic effect on the relationship between the built and the unbuilt.

Architecture has established a hierarchy between the unbuilt and the built that rests on the belief that "untouched" geology is inert. Barren land was thought to be the opposite of civilization. Nonlinear science has shown that geological systems are similar in complexity to civilization. The two systems are driven by the same matter-energy relationships but proceed at much different rates. This concept of time, or duration, becomes an important way to look at actual differences between man and nature. Civilization itself can be viewed as a complex system driven by the matter-energy relationship. From this standpoint, the progress of civilization, including the built world, is essentially a response to this relationship, and it exhibits similarities to other large systems obeying these parameters.

Sedimentary rocks, species, and social classes (and other institutionalized hierarchies) are all historical constructions, the product of definite structure-generating processes that take as their starting point a heterogeneous collection of raw materials (pebbles, genes, roles), homogenize them through a sorting operation, and then consolidate the resulting uniform groupings into a more permanent state. [De Landa, 1000 Years, p62]

Just as nonlinear dynamics have shown that inert matter exhibits levels of complexity that match living matter, they can show that geological systems exhibit the same levels of complexity as civilization. The development of geologic systems mirrors the development of civilization at a much slower rate, and thus the key difference between the built and unbuilt becomes the variable of time. This actual difference is ignored by theorists in favor of perceived philosophical differences.



TODD HIDO  
UNTITLED #2736, 2000  
PACIFICA, CA  
CHROMOGENIC PRINT  
24 X 20", 38 X 30", 48 X 38"

Architectural dogma responds to changes in the understanding of the nature of the world through a process of revision rather than replacement. This is the opposite of the approach we find in the scientific community. Even when architects recognize problems with theory, they respond by modifying existing paradigms rather than by rejecting them altogether. Architects believe in the existence of an architectural truth that has to be discovered. The French Philosopher Gilles Deleuze would argue for a different conception of truth, one that is created rather than discovered.

This idea that truth is not something preexistent that we have to discover, but that it has to be created in every domain, is obvious in the sciences, for example. Even in physics, there is no truth that doesn't presuppose a system of symbols, even if they are only coordinates. There is no truth that doesn't "falsify" established ideas. To say that "truth is created" implies that the production of truth involves a series of operations that amount to working on a material - strictly speaking, a series of falsifications. [Deleuze, "Mediators" p287]

If we choose to reject certain tenets of architectural dogma, based on the weakness of the man versus nature argument, a new set of design options become available, and the direction of the profession changes.

Architecture's relationship with time has previously focused on the concept of monumentality. In many ways, monumental architecture is civilization's fight against the permanence of nature; it becomes a symbol of civilization's conquest of the unbuilt (permanent transformation of unbuilt into built). This emphasis on competition between man and nature rests on misconceptions about the differences between the built and unbuilt. When we view the two systems, geology and civilization, as similar in complexity but differing in duration, monumentality becomes less important as a design concern. In fact, geology lasts so much longer than civilization that any battle against the permanence of nature seems misguided and ignorant. Deleuze explores the concept of duration as it relates to the philosophy of Bergson, and argues for an understanding of time that places less emphasis on the idea of an object enduring.

We must not say that external things endure, but rather that there is some inexpressible reason in them which accounts for our inability to examine them at successive moments of our duration without observing that they have changed." [Deleuze, "Bergsonism" p48]

When we view both the world of civilization and the world of geology as kinetic systems, we can get beyond the idea of architecture as a tool in man's progress against nature. We are prisoners of a very narrow-minded ideal of architecture: one that is concerned with duration and elitism. Perhaps monumentality derives from a fear of death, or the desire for immortality, but it ultimately places emphasis on a certain type of building. A masterpiece must fit the criteria of permanence. We favor the solid and static over the light and kinetic. This seems inappropriate to our contemporary world: one of migrants, rapidly evolving technology, genetic engineering. Our world is becoming more dynamic at an exponential rate, and we are still obsessed with the immobile.



TODD HIDO  
UNTITLED #2904, 2001  
PACIFICA, CA  
CHROMOGENIC PRINT  
24 X 20", 38 X 30", 48 X 38"

Architects are in a rare position. Our work is a unique combination of problem solving and cultural statement: the balancing act of scientific foundation and artistic possibility. Through the combination of these two realms, the limits of architecture are pushed beyond their current state. Here lies the real power of the profession: not to be canonical, but to push the boundaries of man's possibilities. There is no way to reach these true monuments if the scientific foundation of architecture rests on inaccurate assumptions. Remodeling these assumptions is not a valid solution. They must be abandoned. As boundaries are pushed, new possibilities will emerge. The idea is not to find the end, but to find the edge.



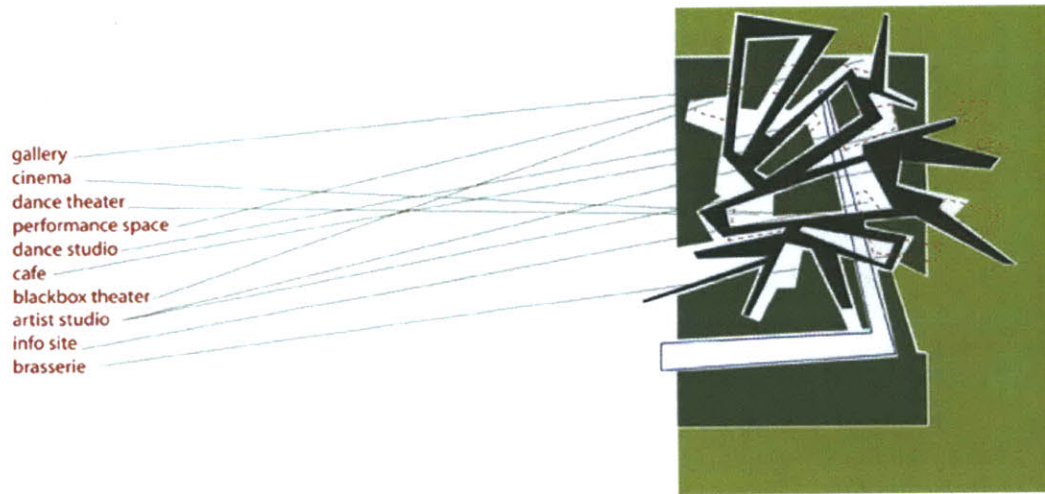


## II. FOUNDATION

FOUNDATION  
DESIGN WORK INFORMED BY DURATION



AN EARLY SKETCH OF THE VARIOUS LAYERS: FOREST FLOOR, FOLIAGE, PATH,



SCHEMATIC OF THE MASTER PLAN FOR THE ENTIRE SITE

## ARCHITECTURAL LAYERS AND RATES OF CHANGE

The design of an Expo master plan and pavilion incorporates an understanding of duration as the primary method for increasing the architecture's connection with time. The proposal is a series of architectural events set in a landscape of isolation and discovery. This is accomplished through a series of structural and compositional layers: the forest floor, a network of elevated paths and ramps, barrier walls, large signage walls, and finally, contained architectural space. An essential element of the design is an intentional blurring of the built and unbuilt aspects of the design. The landscape and architecture are universally viewed as temporal, with tangible compositional shifts at various intervals.

I. GRADUAL/NATURAL: The expo plot was set within a forested region with the intention that the forest would overtake the architecture once the expo finished. The architectural infrastructure would first become a service for the landscape but would eventually decay, leaving only traces of itself.

II. SEASONAL: The visual connections between the sites within the forest depend upon the amount of foliage, which shifts seasonally within the forest. The color and feel of the site as well as the user's perception of the architecture depend on the time of year. In the summer, the sites are secluded and almost impenetrable, whereas in the winter they barely seem disconnected.

To understand the more immediate temporal shifts in the landscape, it becomes necessary to examine the design of a single building site within the master plan. The landscape is split into a series of plots which would be designed for specific cultural spaces. One of these sites is an open-air theater for dance performances. The theater is composed of two major elements: a sign wall separating the theater from the circulation route and a container forming the backstage and operational contents of the theater. This design continues to look at rates of change.

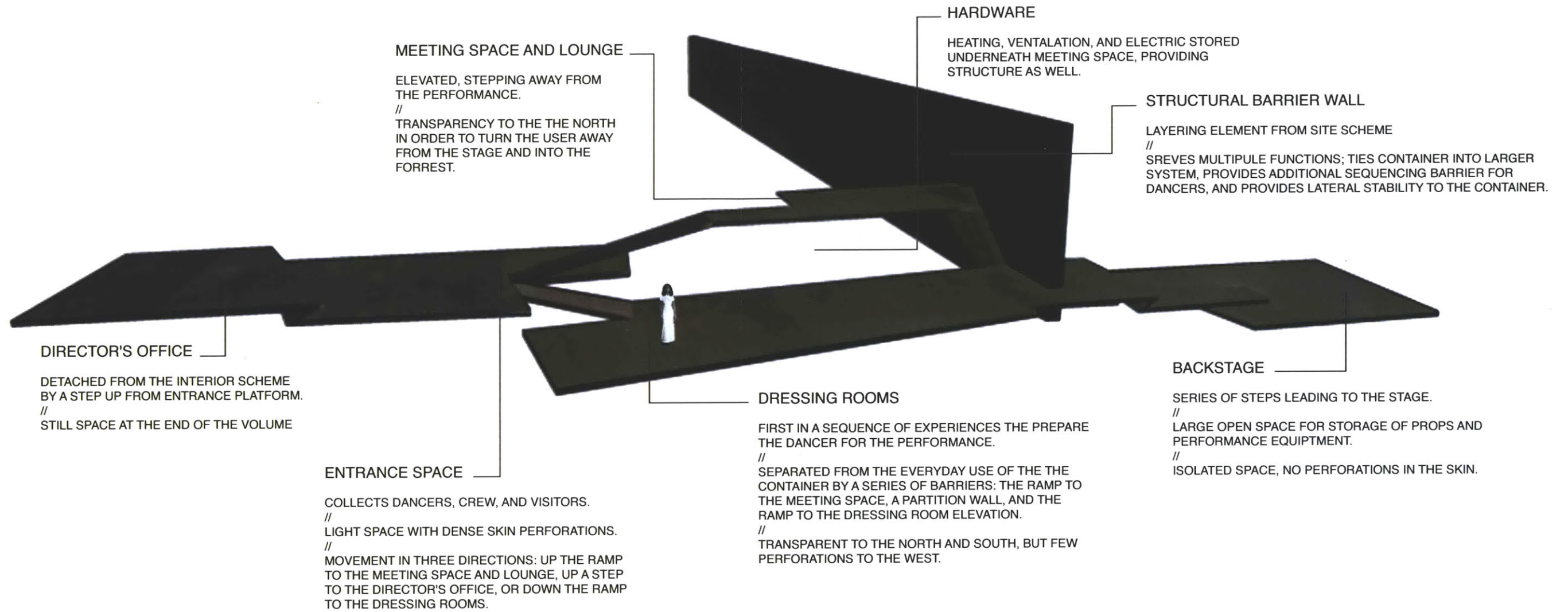
III. DAILY: The container is seen as a solid object with clear borders during the day, but at night it becomes a collection of luminous points within the landscape. The skin of the container is constantly weathering and looks different based on the weather conditions of a certain day.

IV. IMMEDIATE: The surface of the signage wall is constantly changing, expressing the surroundings and displaying projections at various scales. The wall is used to move people through the site and it transforms the constantly changing aspects of the site into visual information at various scales.



**BOARD I:**  
THE MEANING OF PLATFORM

THE MEANING OF PLATFORM: CREATING A SEQUENTIAL EXPERIENCE FOR THE PERFORMERS WHILE PROVIDING PROGRAM FOR THE ENTIRE CONTAINER



**BOARD II:  
THE MEANING OF SKIN**

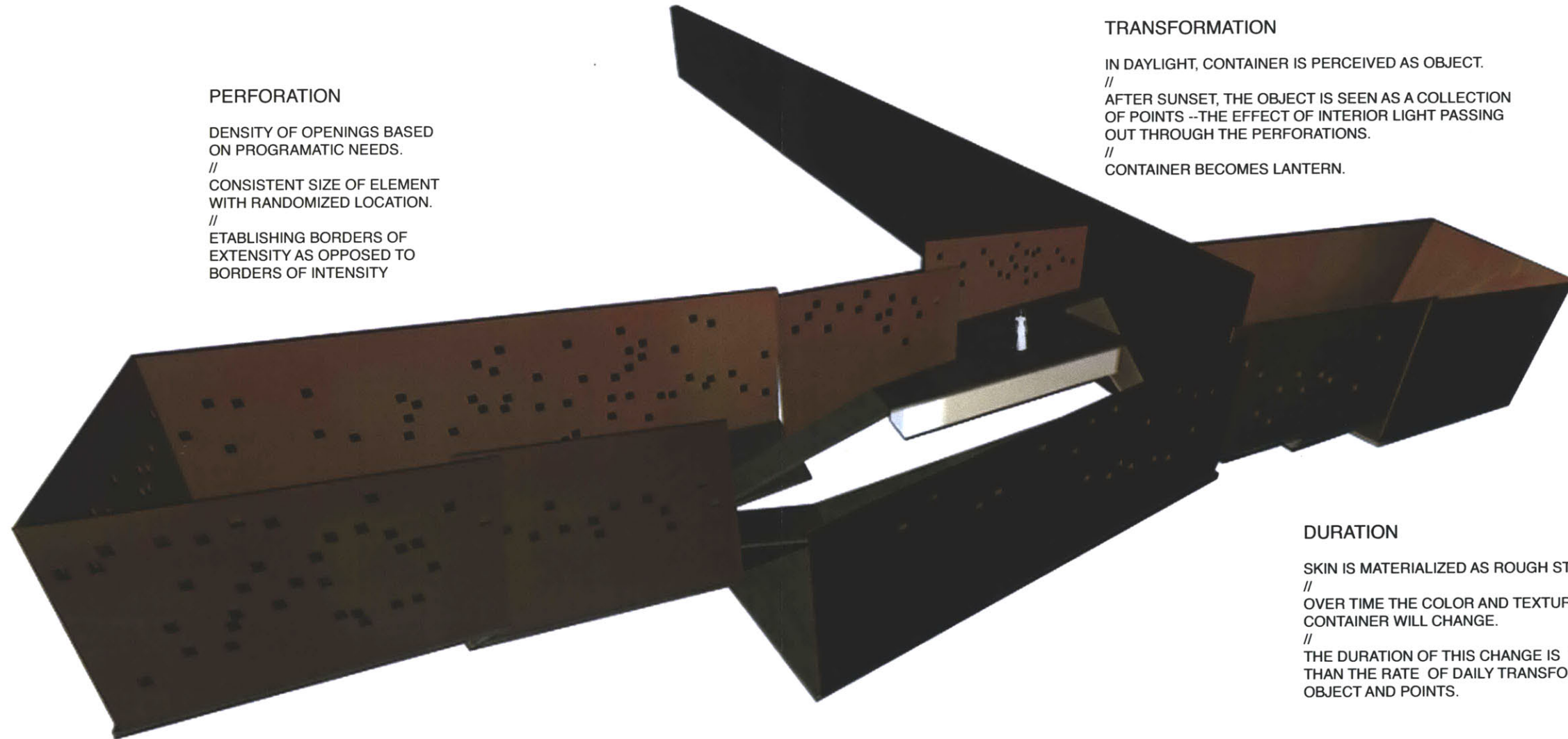
THE MEANING OF SKIN: PROVIDING CONTINUITY AND IDENTITY TO THE CONTAINER WHILE MEDIATING LIGHT BETWEEN INTERIOR AND SITE

PERFORATION

DENSITY OF OPENINGS BASED  
ON PROGRAMATIC NEEDS.  
//  
CONSISTENT SIZE OF ELEMENT  
WITH RANDOMIZED LOCATION.  
//  
ESTABLISHING BORDERS OF  
EXTENSITY AS OPPOSED TO  
BORDERS OF INTENSITY

TRANSFORMATION

IN DAYLIGHT, CONTAINER IS PERCEIVED AS OBJECT.  
//  
AFTER SUNSET, THE OBJECT IS SEEN AS A COLLECTION  
OF POINTS --THE EFFECT OF INTERIOR LIGHT PASSING  
OUT THROUGH THE PERFORATIONS.  
//  
CONTAINER BECOMES LANTERN.



DURATION

SKIN IS MATERIALIZED AS ROUGH STEEL PLATES.  
//  
OVER TIME THE COLOR AND TEXTURE OF THE  
CONTAINER WILL CHANGE.  
//  
THE DURATION OF THIS CHANGE IS MUCH GREATER  
THAN THE RATE OF DAILY TRANSFORMATION BETWEEN  
OBJECT AND POINTS.



**BOARD III:**  
**THE MEANING OF CONTAINER**

THE MEANING OF CONTAINER: DISTINGUISHING ARCHITECTURE FROM ARCHITECTURAL LANDSCAPE

SITE ORGANISM

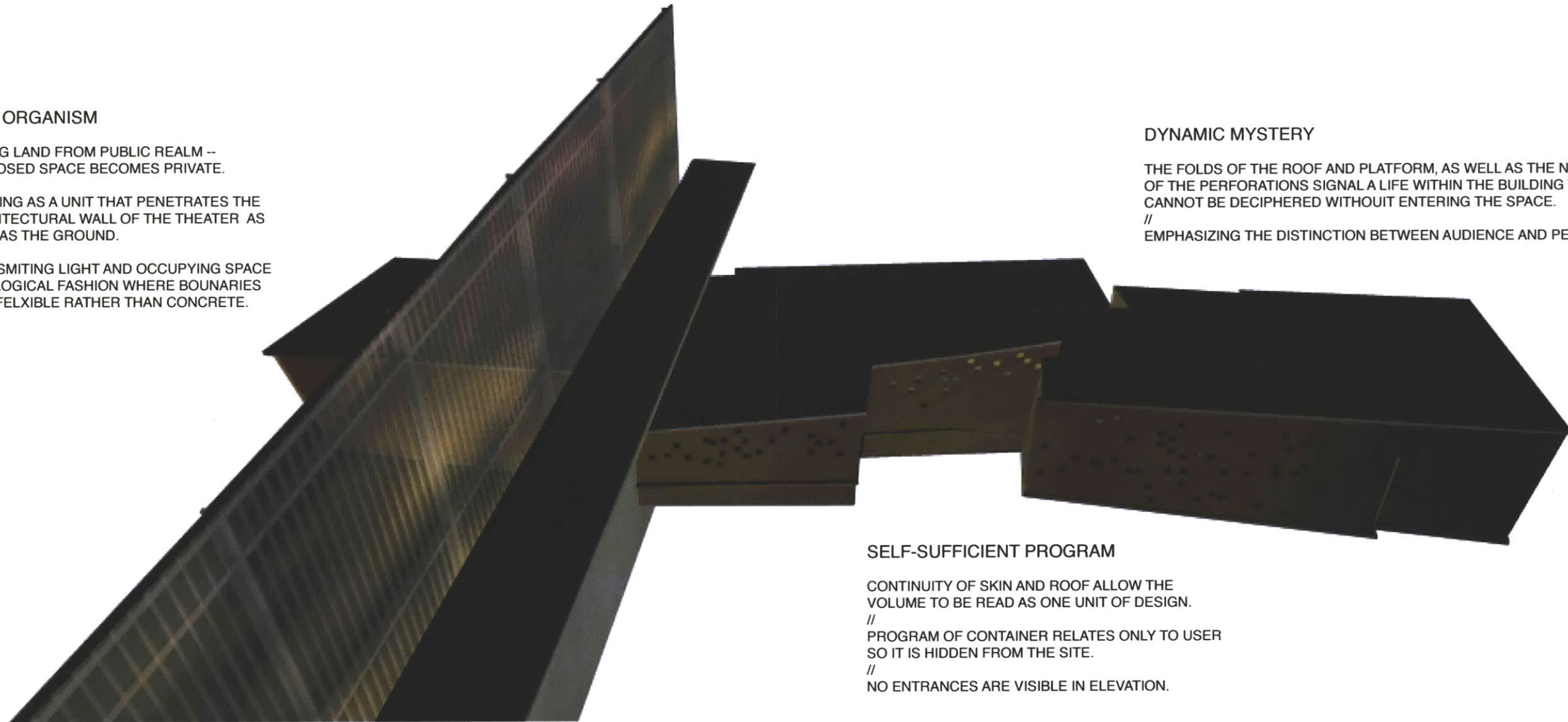
TAKING LAND FROM PUBLIC REALM --  
ENCLOSED SPACE BECOMES PRIVATE.

//

EXISTING AS A UNIT THAT PENETRATES THE  
ARCHITECTURAL WALL OF THE THEATER AS  
WELL AS THE GROUND.

//

TRANSMITTING LIGHT AND OCCUPYING SPACE  
IN BIOLOGICAL FASHION WHERE BOUNDARIES  
ARE FLEXIBLE RATHER THAN CONCRETE.



DYNAMIC MYSTERY

THE FOLDS OF THE ROOF AND PLATFORM, AS WELL AS THE NATURE  
OF THE PERFORATIONS SIGNAL A LIFE WITHIN THE BUILDING THAT  
CANNOT BE DECIPHERED WITHOUT ENTERING THE SPACE.

//

EMPHASIZING THE DISTINCTION BETWEEN AUDIENCE AND PERFORMERS,

SELF-SUFFICIENT PROGRAM

CONTINUITY OF SKIN AND ROOF ALLOW THE  
VOLUME TO BE READ AS ONE UNIT OF DESIGN.

//

PROGRAM OF CONTAINER RELATES ONLY TO USER  
SO IT IS HIDDEN FROM THE SITE.

//

NO ENTRANCES ARE VISIBLE IN ELEVATION.

**BOARD IV:**  
**THE MEANING OF WALL**

THE MEANING OF WALL: SURFACE OF INFORMATION AT LEVELS OF SCALE AND ABSTRACTION

ENVIRONMENTAL ABSTRACTION

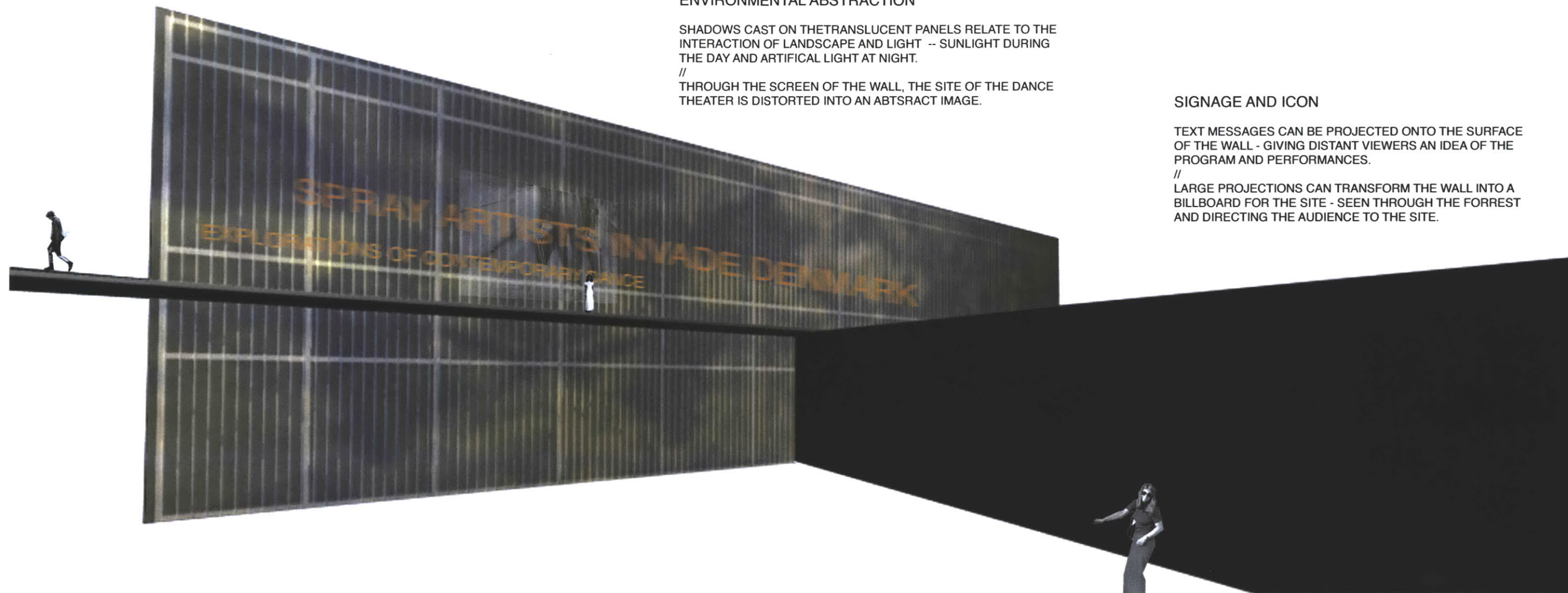
SHADOWS CAST ON THE TRANSLUCENT PANELS RELATE TO THE INTERACTION OF LANDSCAPE AND LIGHT -- SUNLIGHT DURING THE DAY AND ARTIFICIAL LIGHT AT NIGHT.

//  
THROUGH THE SCREEN OF THE WALL, THE SITE OF THE DANCE THEATER IS DISTORTED INTO AN ABSTRACT IMAGE.

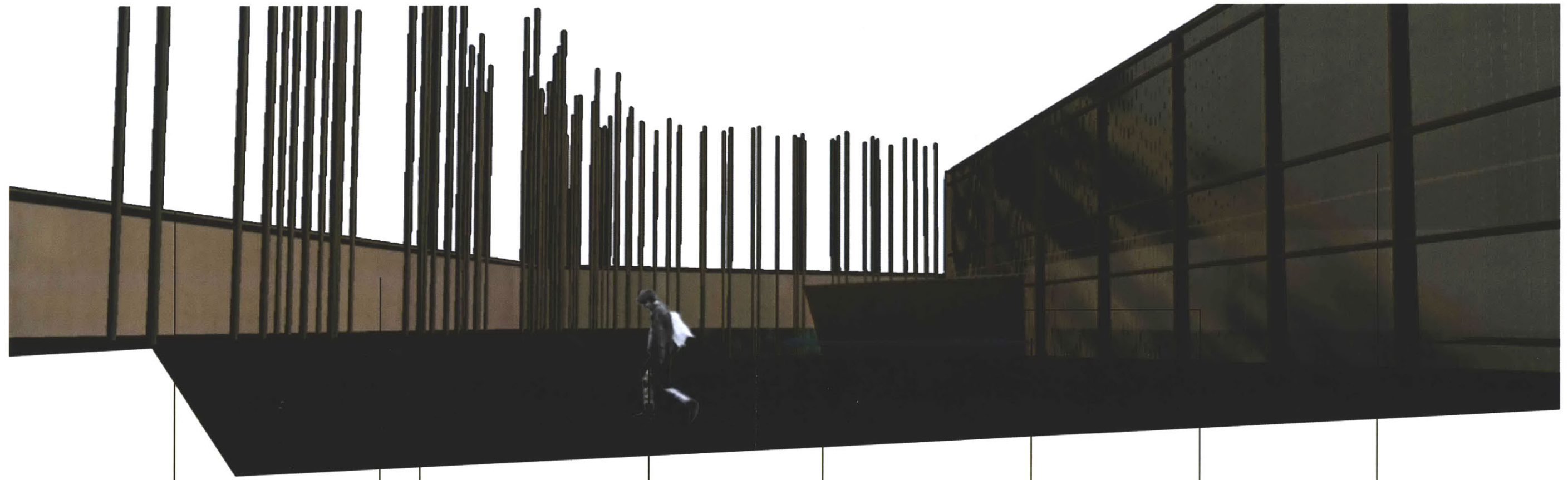
SIGNAGE AND ICON

TEXT MESSAGES CAN BE PROJECTED ONTO THE SURFACE OF THE WALL - GIVING DISTANT VIEWERS AN IDEA OF THE PROGRAM AND PERFORMANCES.

//  
LARGE PROJECTIONS CAN TRANSFORM THE WALL INTO A BILLBOARD FOR THE SITE - SEEN THROUGH THE FOREST AND DIRECTING THE AUDIENCE TO THE SITE.



**BOARD V:**  
THE MEANING OF LANDSCAPE



PATH

EMPHASIZES MOVEMENT CONNECTION, AND CLARITY: ELEVATED ABOVE PROGRAM TO LINK EXPO PLOTS.

BARRIER WALLS

DIVIDES PROGRAM OF EXPO PLOTS INTO INTIMATE REGIONS.

LIGHT COLUMNS

SUPPORTS CANOPY STRUCTURE OF DANCE THEATER AND SERVES AS A SLOW DEVICE: PAUSING VISITORS AT THE SITE.

FORREST FLOOR

CHANGES IN ELEVATION IN RESPONSE TO PROGRAM.

ASPHALT SURFACE

ARTIFICIAL GROUND DISCOVERED BENEATH SURFACE OF FORREST FLOOR.

STAGE

PERFORMANCE REGION AND LINK BETWEEN PUBLIC AND PRIVATE USE.

CONTAINER

PRIVATE NEEDS DEFINE THE ENCLOSED STRUCTURE.

MEDIA WALL

DISPLAYING SITE INFORMATION AT LEVELS OF SCALE AND ABSTRACTION.

THE MEANING OF LANDSCAPE: SYSTEMATIC LAYERING TO PROVIDE ARCHITECTURAL SPACES WITHOUT FIXED BOUNDARIES

### III. COMPUTATIONAL APPLETS

**ADAPTIVE AND EVOLUTIONARY FORM**  
ENABLING ARCHITECTURE TO FUNDAMENTALLY CHANGE WITH TIME



## GROWTH AND DECAY OF PHYSICAL SPACE

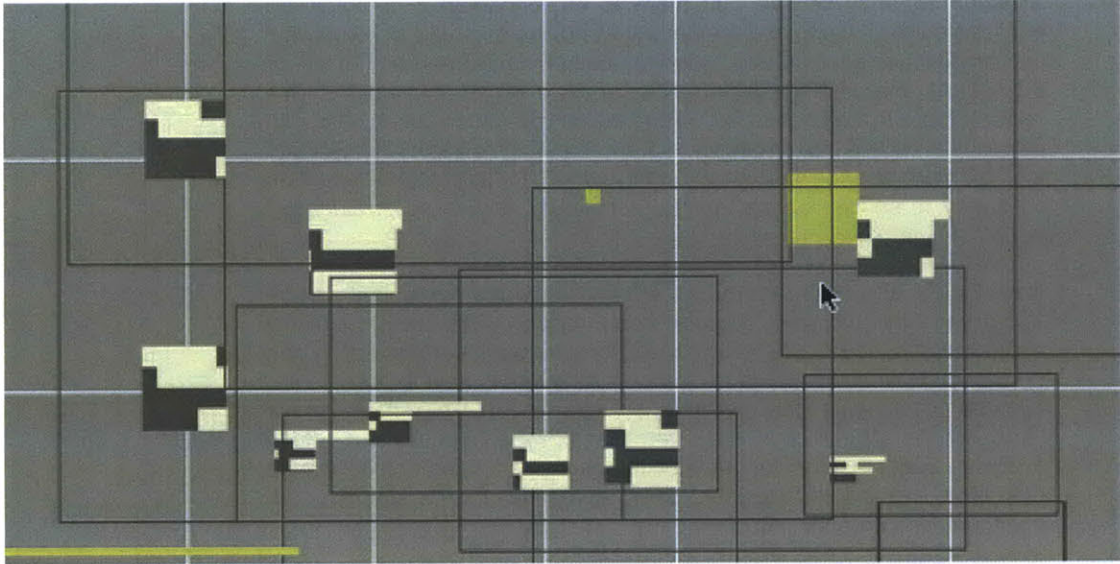
Although we may believe that there is no fundamental contradiction between the built world and the natural world, we must still acknowledge that architecture's relationship with time has primarily been one of aging. Essentially the built world deteriorates over a range of durations. It's relationship with time is subtractive and the only way to counteract the role of time is to rebuild the building or to add something new to it. Rarely does the built world change through its own initiative. Today computation offers us a way to enable architecture to reconfigure itself over an extended period of time.

If we look to simple organisms we will find a natural model that can be applied to the built world through computation. This thesis primarily explores the concepts of short-term and long-term adaptation as they can be applied to architectural form. When a structure is able to balance immediate adaptation and its own internal history, it can exhibit properties of biological life. At this point, the differences between artificial and natural form truly begin to dissolve. Architecture begins to fundamentally change over time, and architectural complexity approaches the complexity of the environment surrounding it.

Architects have looked to both kinetic systems to answer questions regarding short-term adaptation in architecture and genetic algorithms as a technique for long-term adaptation, but they have yet to synthesize these two ideas. Kinetic systems allow physical space to change dramatically post-construction, but these systems lack an internal history and there is a corresponding lack of distinction between the states of the system. Genetic properties have been used to model and generate form, but they have yet to be applied to a building post-construction in order to enable space to evolve over the course of a building's life.

This thesis aims to look at the possibilities of evolutionary form, first as diagrammatic sketches, and then as architectural surface. The initial work focuses on gaining an understanding of genetic and adaptive algorithms as a visual and spatial phenomenon. It soon becomes clear that the goal of the work is not to formulate a literal genome for a physical space, but instead to bring the ideas of history and memory to built form. The work develops into a balancing of short-term and long-term adaptive traits in a kinetic surface such that the resulting space develops a distinct character over time -- one that changes as it is used in different ways and by different users.

COMPUTATIONAL APPLETS  
SKETCHES OF ADAPTIVE AND EVOLUTIONARY FORM



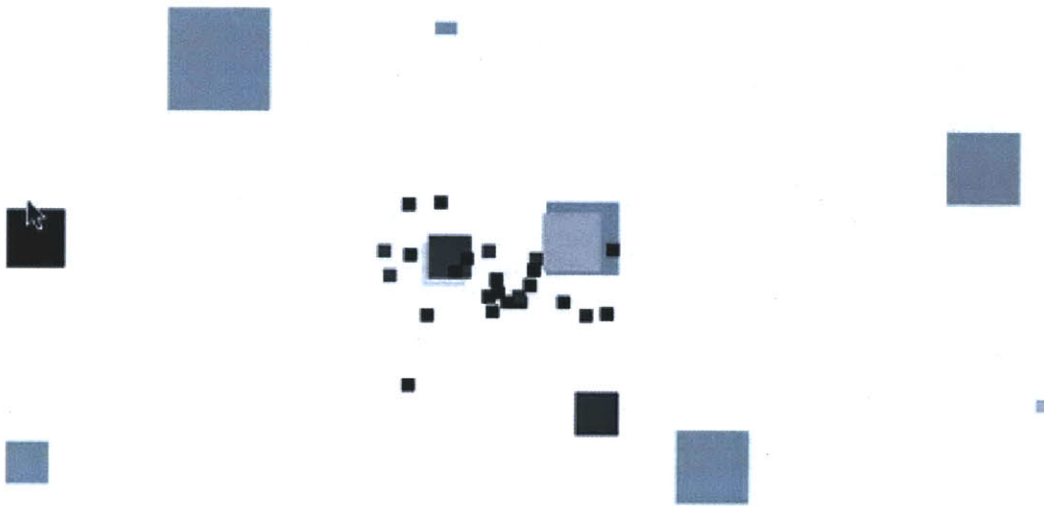
## MODELING BIOLOGICAL PROPERTIES AS ARCHITECTURAL SKETCH

This work is concerned with the accumulation of memory in a computational system based on properties of biological life. In order to better understand the spatial and visual consequences of biological systems, the properties of these systems are modeled as visual sketches. These Java applets, either adaptive or evolutionary in character, provide a conceptual foundation for the design of a computational architectural surface with memory.

**ADAPTIVE SYSTEM:** The adaptive system consists of a feedback loop and an internal history that develops over time. As the user interacts with the system, the system responds, but in ways that change based on previous user interaction with the system. This system is composed of simple organisms that have a life cycle where their survival is based on fitness criteria. These organisms affect the entire system, but they do not reproduce and pass their genetic material to their offspring.

**EVOLUTIONARY SYSTEM:** The evolutionary system is based on the properties of genetics and the algorithms involved in natural selection. Organisms with a genetic code survive and reproduce based on certain selective and fitness requirements. When two organisms reproduce, genetic crossover and mutation determine the genes the offspring receive. The most fit organisms are surviving the longest and have the greatest chance of reproducing. In this way, the population is always moving towards an optimal configuration -- one determined by user interaction with the system.

Both of these systems provide ways to balance short-term and long-term adaptation. The user is able to directly effect the system by interacting with the applet, but the result of the interaction depends on the internal history of the system. Adaptive and evolutionary models become the vehicle for accomplishing this accumulation of memory. In the end however, the emphasis is on memory rather than the algorithms used to accomplish this memory.



```
//life cycle of blocks regulated by user input  
//size and position of blocks determine fitness  
//blocks grow if clicked and can be dragged to new locations  
//above a certain threshold, blocks are anchored  
//anchored blocks get agitated if clicked  
//anchored state determines where new blocks will be born  
//nothing lasts forever ...
```

## ADAPTIVE APPLET no. 002

<http://web.mit.edu/jroth/www/thesis/>

This sketch consists of a series of blocks that can be altered by user input. Blocks have a life cycle where their fitness is based on their position and size. By changing the sizes and positions of the blocks, the user also changes how new blocks are added to the system. The visual patterns that emerge are based on a combination of user input and internal history, but the system is not designed to evolve. This sketch is primarily concerned with the nature of feedback in the system, and examines the indirect consequences of user interaction.

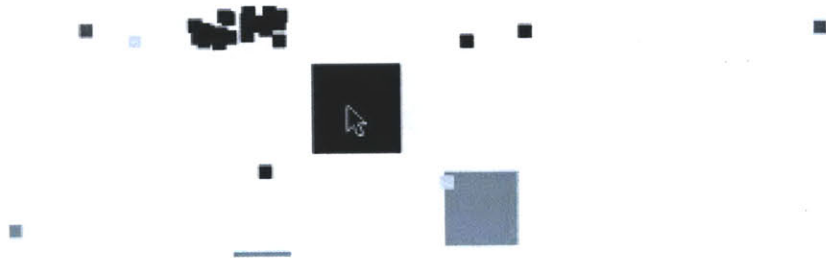
**SELECTION ALGORITHM:** Blocks have a “life ability” that determines their probability of surviving to the generation of the program. This life ability is based on their distance from the place of their birth as well as their size and age.

```
public void tallyLife() {
    if(!anchored) {
        life_ability = 0
        //distance from the lifeline:
        distance = Math.abs(y_val-175);
        more = distance-20;
        if (distance>20) life_ability += 2*distance/3;
        //age factor:
        life_ability += 40-generation;
        //size factor:
        if (size>16 && size <= 52) life_ability += size;
        f(life_ability>=100) {
            anchored = true;
            anchor_clock = 60;
        }
    }
    if(anchored) anchor_clock -= 1;
}

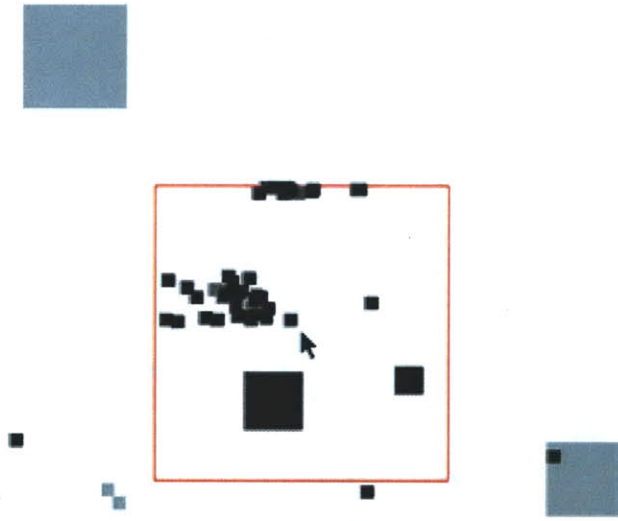
public void killingSpree() {
    if(!anchored) {
        int hit = r.randomInt(100);
        if (life_ability<5) {
            if (hit>40) alive = false; }
        if (life_ability>5 && life_ability<20) {
            if (hit>70) alive = false; }
        if (life_ability>20 && life_ability<60) {
            if(hit>90) alive = false; }
        if (life_ability>60 && life_ability<100) {
            if(hit>97) alive = false; }
        }
    else {
        if(anchor_clock < 0) alive = false;
    }
}
```



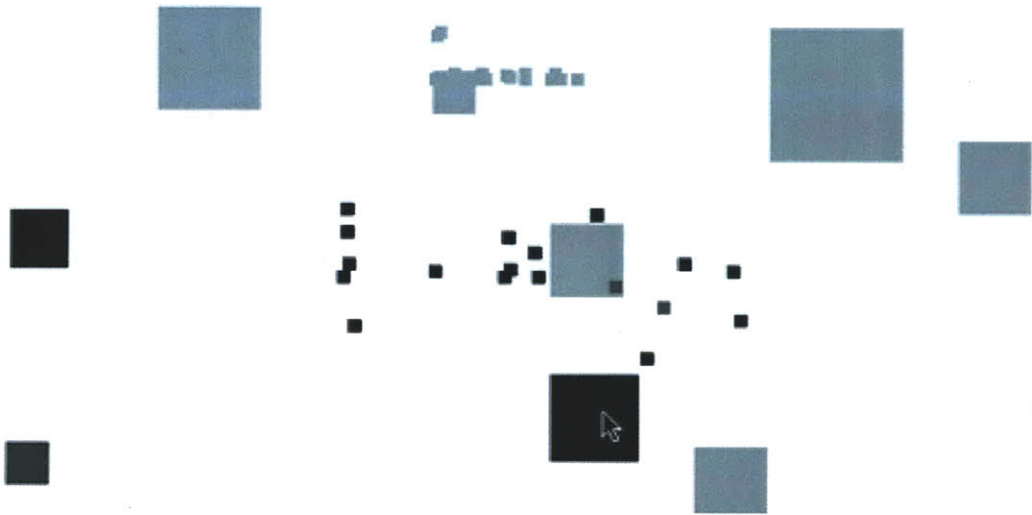
THE BLOCKS ARE INITIALLY DISPERSED OVER THE ENTIRE HORIZONTAL



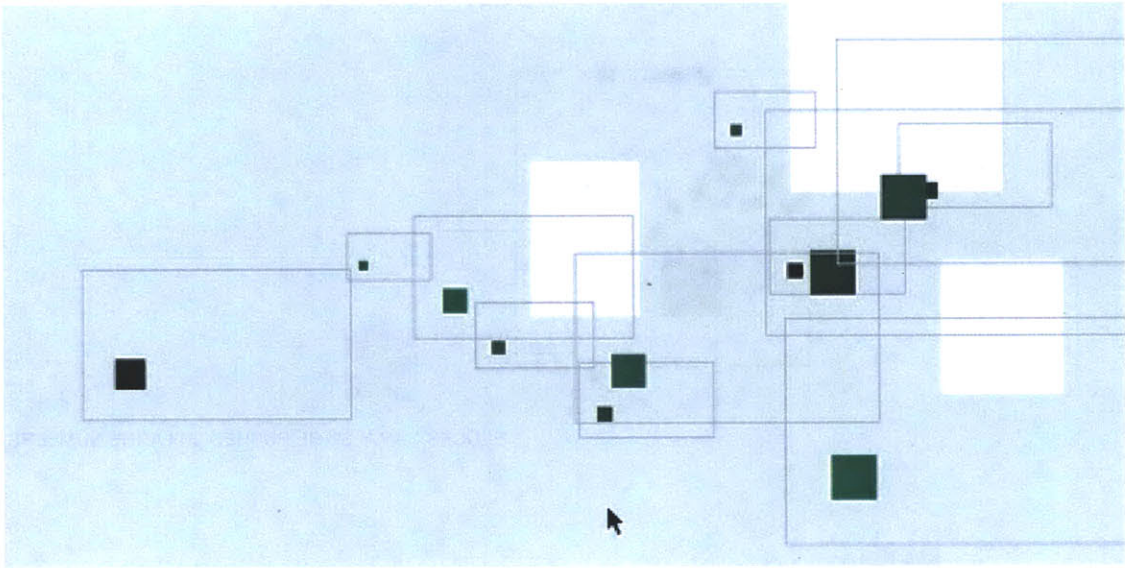
AS THE USER SELECTS BLOCKS AND BRINGS THEM TO NEW LOCATIONS, THE NEW STATE OF THE SYSTEM DETERMINES WHERE THE NEXT GENERATION OF BLOCKS WILL BE BORN.



BLOCKS CAN ALSO BE PUSHED IN LARGE NUMBERS



AS BLOCKS PASS A THRESHOLD IN LIFE ABILITY, THEY BECOME ANCHORED IN POSITION AND GREY IN COLOR. THESE ANCHORED BLOCKS DETERMINE THE STATE OF THE SYSTEM. THE ORGANISMS BECOME THE CONTROLS. CLICKING ON ANCHORED BLOCKS WILL AGITATE THEM AND CHANGE THE VERTICAL DISPERSION OF THE NEXT GENERATION OF ORGANISMS. TEMPORARY VISUAL PATTERNS EMERGE AND GENERATE NEW PATTERNS.



```
//organisms are born, exist, and die in dataspace  
//user input defines the environment  
//organisms act in response to the environment  
//selective criteria determines who will reproduce  
//genetic code of organisms is passed to offspring  
//variation through crossover and mutation  
//form emerges
```



## EVOLUTIONARY APPLET no. 001

<http://web.mit.edu/jroth/www/thesis/>

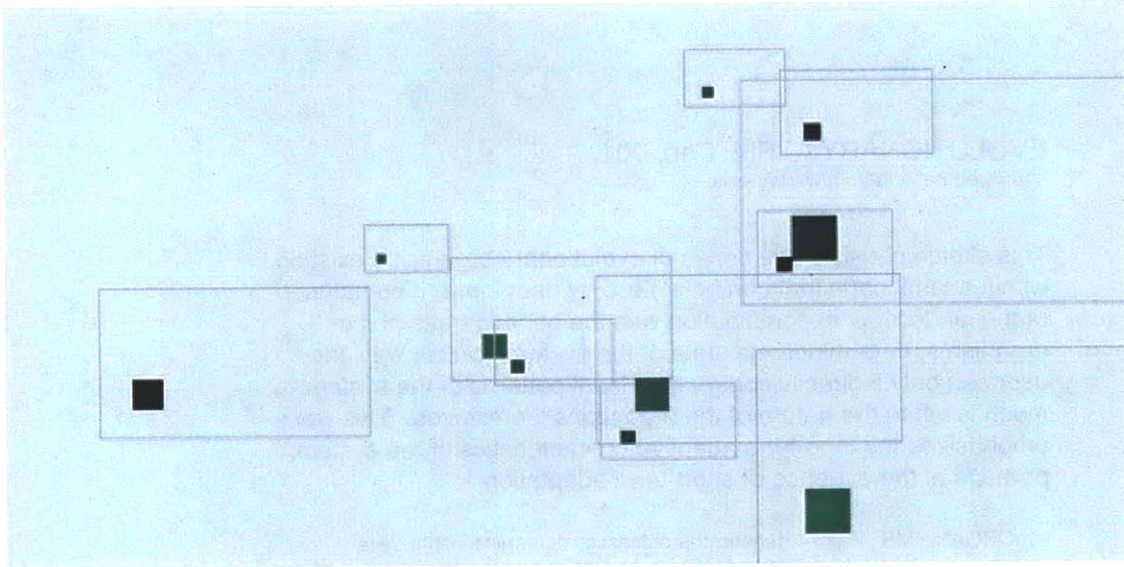
This sketch consists of a series of evolutionary organisms existing within a landscape that can be altered by user input. The nature of the landscape, in combination with the genetic traits of the organisms, determines the state of the system. In this way, the user can only indirectly control the visual patterns of the system -- much is left to the nature of the organisms themselves. This work emphasizes the long-term adaptive characteristics of the system, perhaps at the expense of short-term adaptation.

evoORGANISMS:        genetic characteristics determine visual traits  
                         reproduction based on selective criteria  
                         offspring inherit parental genetic information

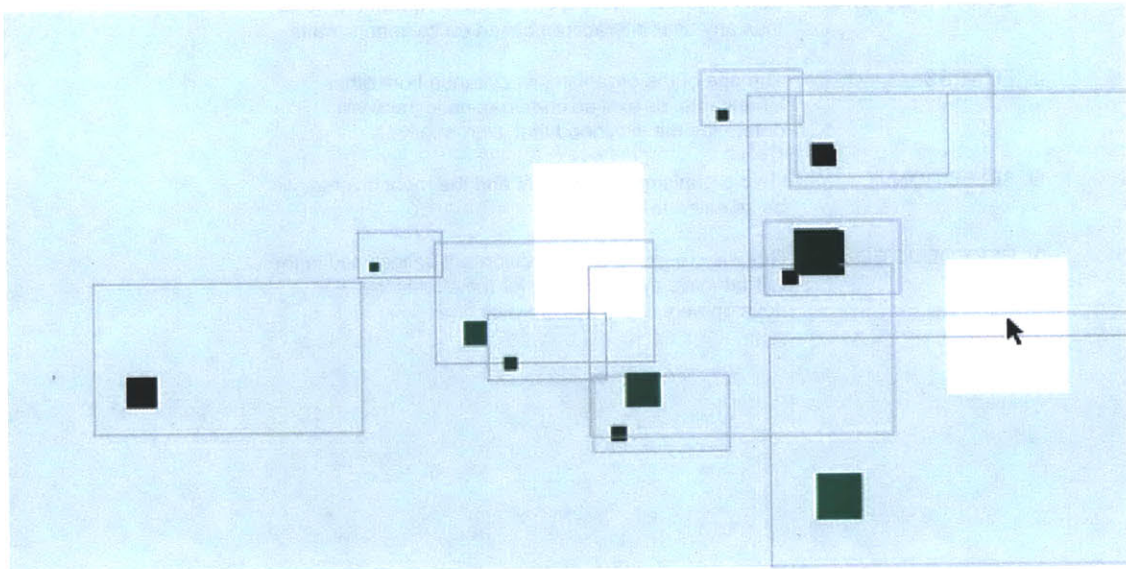
dataSPACE:            landscape governing the life cycle of evoOrganisms  
                         provides rules governing selection  
                         user input directly effects the landscape

### GENETIC ALGORITHM:

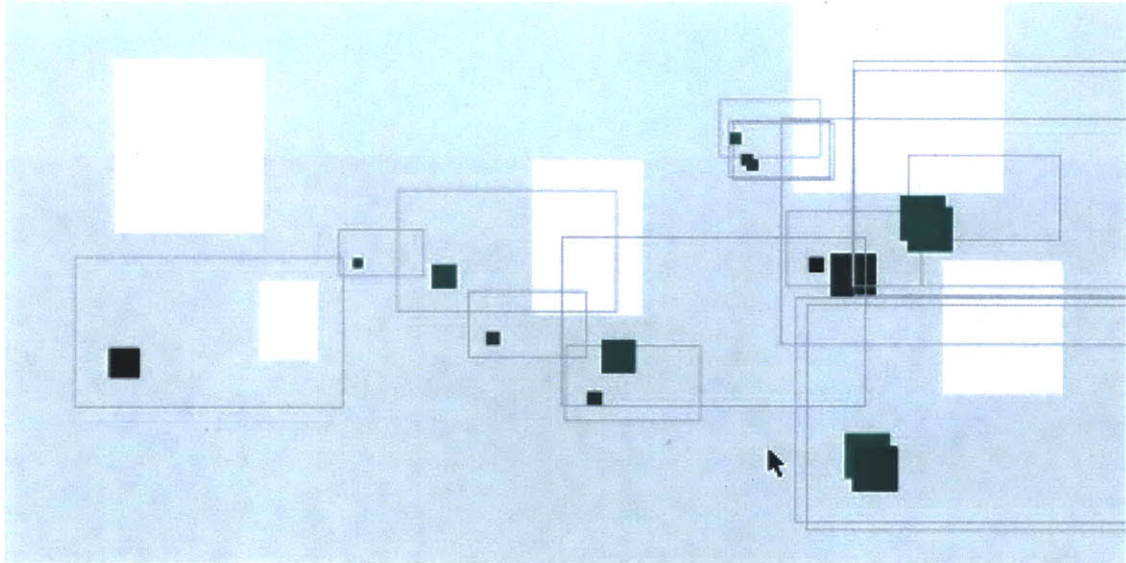
- I. ACTION / RESPONSE: Each organism will respond to the environment (and thus any user interaction) based on its genetic traits.
- II. FITNESS:            The age of the organism, its distance from other inhabitants, as well as certain genetic traits will determine the likelihood that it will survive.
- III. SELECTION:        Two organisms, the most fit and the most diverse, will be selected to reproduce.
- IV. REPRODUCTION:   Genetic crossover and mutation will be included in the reproductive cycle to allow for the evolutionary development of the system over time.



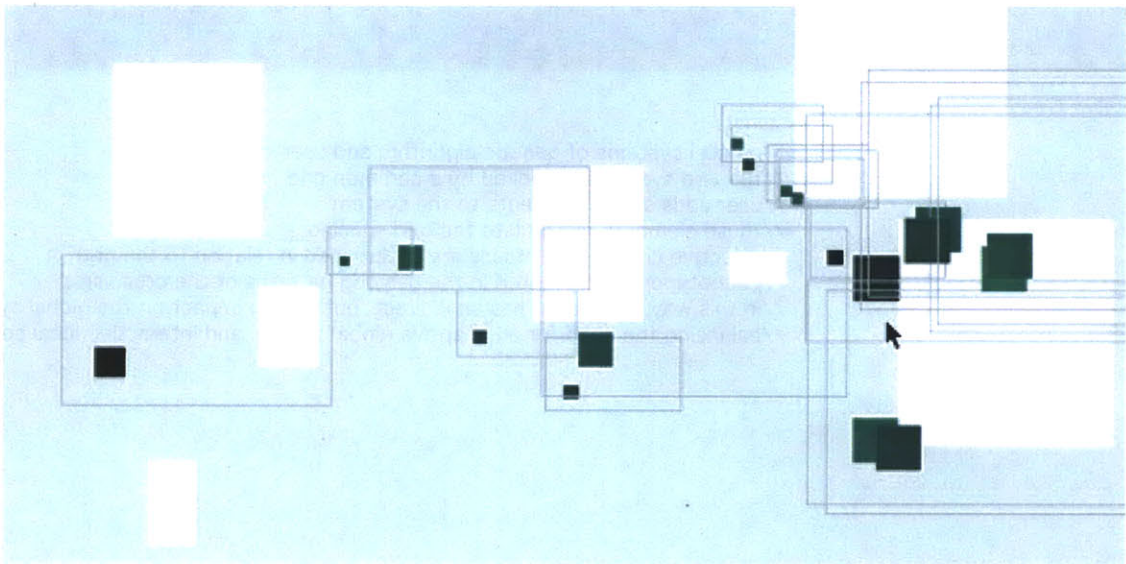
A POPULATION OF ORGANISMS IS RANDOMLY GENERATED AS THE APPLLET OPENS



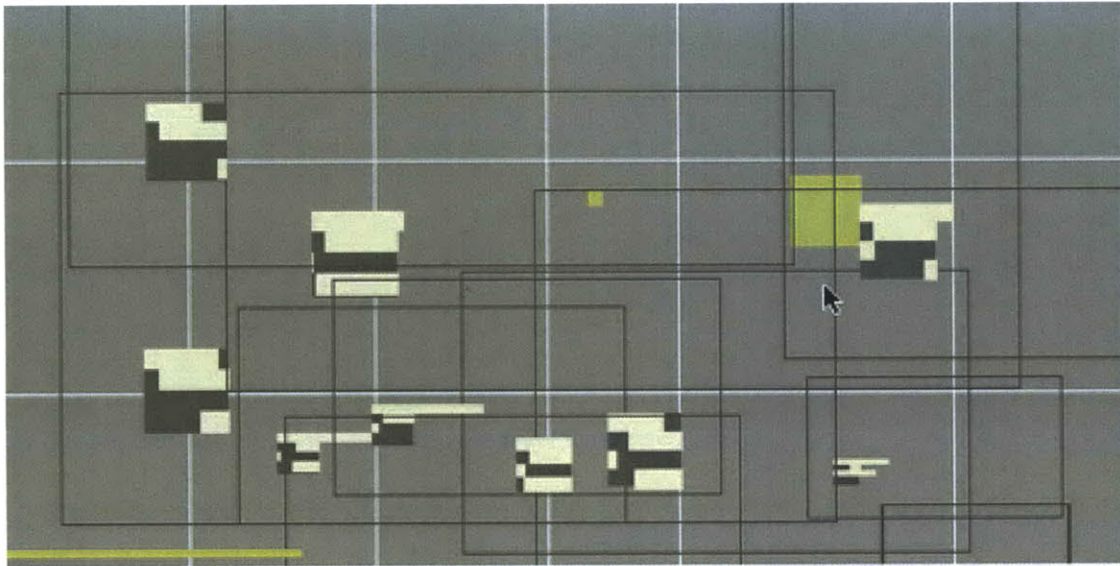
THE ORGANISMS IMMEDIATELY BEGIN TO MOVE, REPRODUCE, AND POSSIBLY DIE. THE USER CAN EFFECT THE SYSTEM BY CLICKING ON THE LANDSCAPE AND CREATING POCKETS OF UNINHABITABLE SPACE. THE LOCATION OF ORGANISMS RELATIVE TO THESE POCKETS BEGINS TO EFFECT THE LIKELIHOOD THAT THEY WILL REPRODUCE.



THE SYSTEM BEGINS TO CONVERGE AT A STATE INFORMED BY USER INPUT



GENETIC ALGORITHMS ARE PRIMARILY USED FOR OPTIMIZATION AS THEY TEND TO MOVE TOWARDS A SPECIFIC CONFIGURATION. IN THIS CASE, THE BEST ORGANISMS ARE CONTINUOUSLY REPRODUCING AND THE VISUAL PATTERN BEGINS TO BECOME CONGESTED AT SPECIFIC POINTS AND EMPTY AT OTHERS.



```
//parallel systems of genetic algorithm and user input  
//the two systems are linked by a common grid  
//user adds simple elements to the system  
//these elements manipulate the grid spacing  
//selective criteria and fitness are determined in relation to the grid  
//phenotype is represented in the banding patterns of the organisms  
//in this way, user input has an indirect, but lasting impact on the global system  
//balancing the need for an adaptive global system and interactive local control
```

## EVOLUTIONARY APPLET no. 002

<http://web.mit.edu/jroth/www/thesis/>

This applet focuses on adding elements of stability to previous sketches. The system uses a grid to keep populations of inhabitants stable. Genetic characteristics of the organisms are also made more visible through a series of graphic bands. The user interacts with the system by adding simple organisms to the landscape. These organisms have the ability to change the position of the grid, and as the grid changes, new visual patterns emerge. Essentially the grid becomes a mediating device between the user and the evolutionary system. In this way, the applet develops the most sophisticated relationship between the system and the user. At this point the evolutionary algorithm reaches a point where it can be applied to architectural form.

**evoORGANISMS:** genetic characteristics determine physical form  
reproduction is based on selective criteria  
offspring inherit parental genetic information

**dataSPACE:** grid space governing the life cycle of evoOrganisms  
Intermediary between user and evoOrganisms  
provides rules governing selection

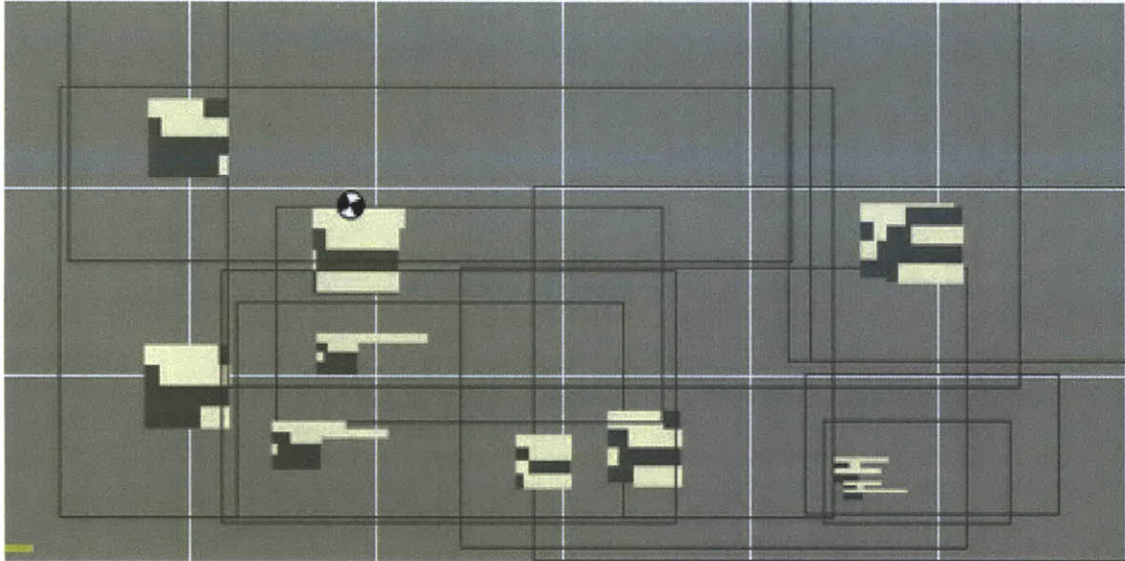
**simpleORGANISMS:** placed into the system by the user  
fixed life span and no reproduction  
collide with grid lines and change gridSpace

### evoORGANISM:

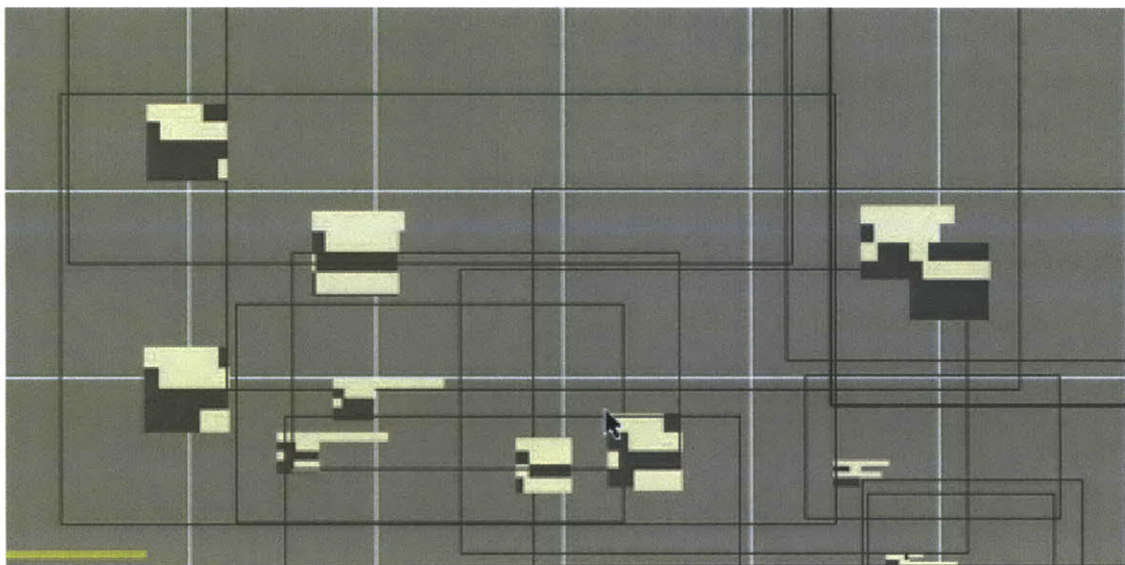
**FOUR GRAPHIC BANDS:** physical representation of genomic data  
**EXTERIOR BOUNDING BOX:** device for observing system patterns  
**SIX GENES:** x\_position, y\_position, size, amplitude, mobility, and color  
**SEXINESS:** factor determining the likelihood of reproduction  
**LIFE ABILITY:** probability that the organism will continue to live

### simpleORGANISM:

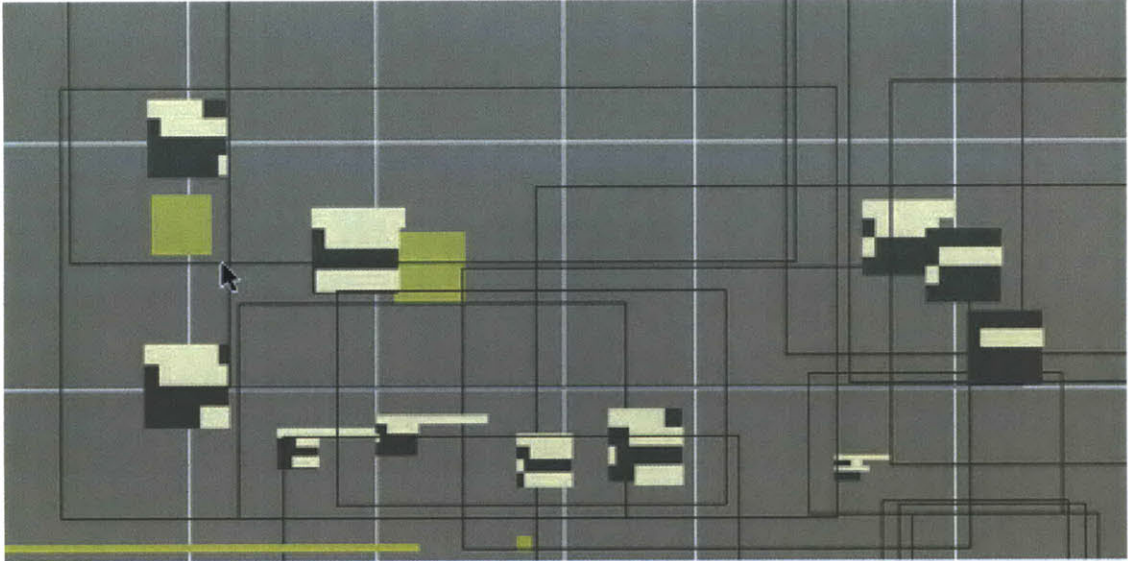
**SIZE:** how much the organism will affect the grid structure of the dataSpace  
**SPEED:** how fast the organism moves  
**LIFESPAN:** how long it will live



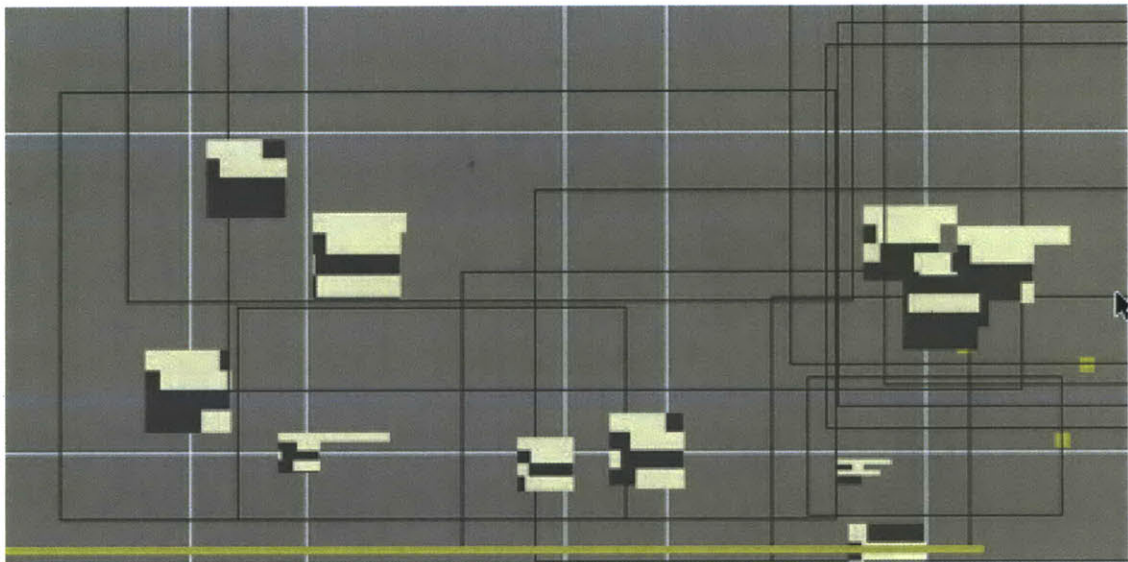
A POPULATION OF ORGANISMS IS RANDOMLY GENERATED AS THE APPLLET OPENS



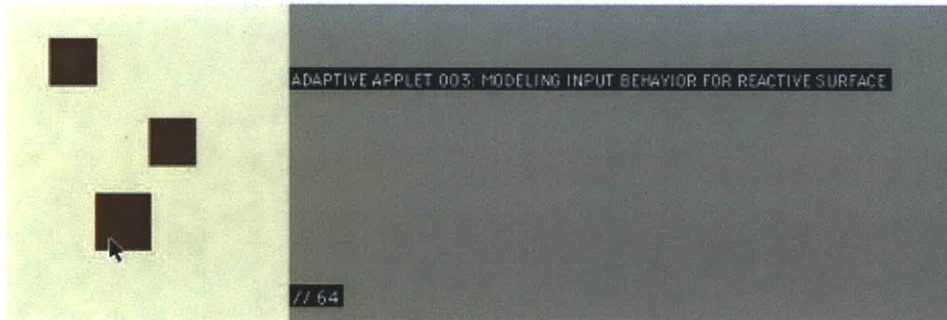
AS THE SYSTEM DEVELOPS, THE GRAPHIC BANDING OF THE ORGANISMS ALLOWS THE CYCLE OF REPRODUCTION AND MUTATION TO BECOME MORE VISUALLY OBVIOUS. THE GRID-DEFINED SELECTION AND FITNESS RULES PREVENT THE SYSTEM FROM CONVERGING IN A SINGULAR DIRECTION.



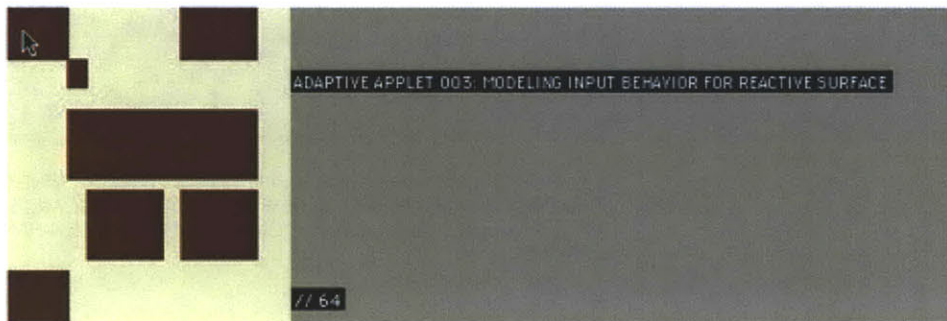
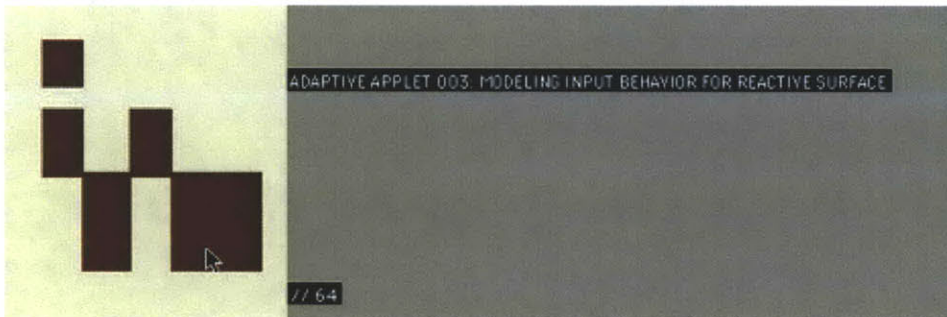
THE USER ADDS THE GREEN SIMPLE ORGANISMS TO THE SYSTEM



THESE ORGANISMS CHANGE THE POSITION OF THE GRID LINES WHICH, IN TURN, LEAD THE SYSTEM TO DEVELOP NEW PATTERNS. THE BALANCE BETWEEN SHORT-TERM AND LONG-TERM ADAPTATION IS THE MOST EVEN-HANDED OF ALL THE SKETCHES.



//returning to issues of interaction and adaptation  
//grid manipulation is responsive and cumulative  
//certain actions cannot be undone

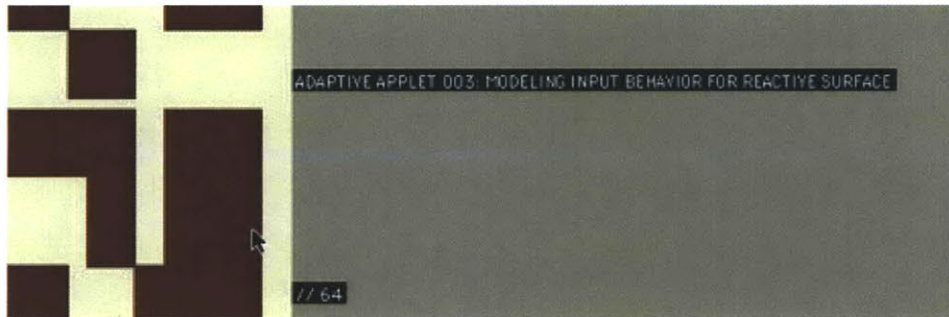




## ADAPTIVE GRID no. 001

<http://web.mit.edu/jroth/www/thesis/>

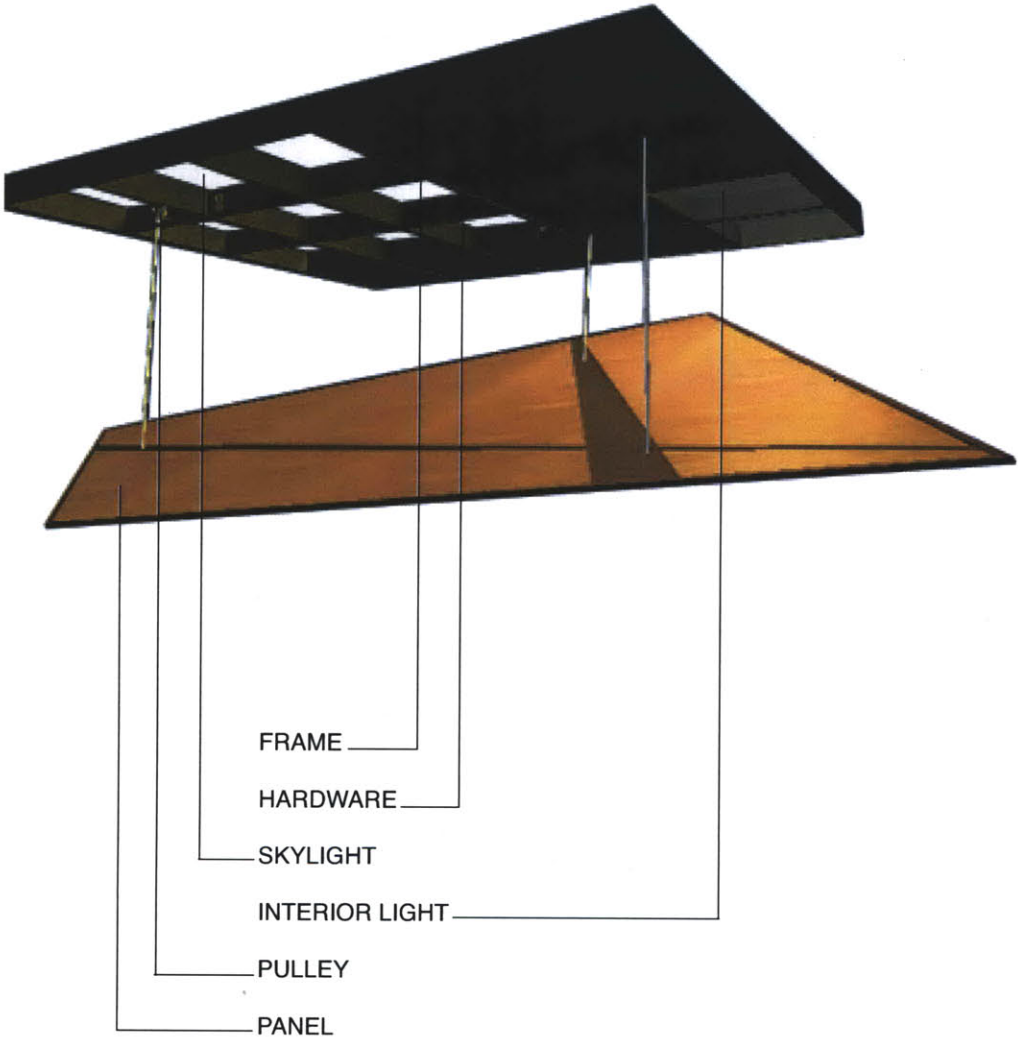
This sketch returns to early ideas of adaptation in order to find a simple algorithm that can be applied to an architectural surface. The complexity of the evolutionary applets seems appropriate to their format as visualizations, but might not necessarily be advantageous as an architectural device. In fact, most of the evolutionary properties of the second applet were artificially constrained in order to prevent it from converging on an optimal solution. This grid is the simplest in terms of computation, but it balances short-term and long-term adaptation more elegantly than previous work. The system changes through growth and decay. Visually, the grid helps the user see the system as one organism composed of a collection of elements. The design of an architectural surface is the obvious next step.





#### IV. ADAPTIVE SURFACE

ADAPTIVE SURFACE  
MOVEMENT AND MEMORY



## COMPUTATIONAL ARCHITECTURES OF TIME

The emphasis of this thesis is the development of a true architecture of time, one that establishes its own internal history and memory. This work includes the design of a computational architectural surface composed of individual units of skylight, interior light, and kinetic panel. Each unit functions as a light modulator and the units can be networked to form an aggregate computational surface.

This surface has the ability to sculpt space through the physical movements of each panel. Three pulleys connected to steel cables manipulate the height and rotation of the wood panel. These panels respond to direct user input and the surface also modulates according to a variety of ambient sensing. The collective surface develops a history that is interconnected and cumulative. In this way, the physical architecture displays characteristics of short-term and long-term adaptation.

The panels occupy the space above the inhabitants and modulate the architecture in subtle, but effective ways. Manipulating spatial and optical qualities in collaboration allows these computational units to have a profound effect on the experience of a space. Such a place seems to be inhabited by a living architecture, one that responds to the needs of the user while retaining its own identity.

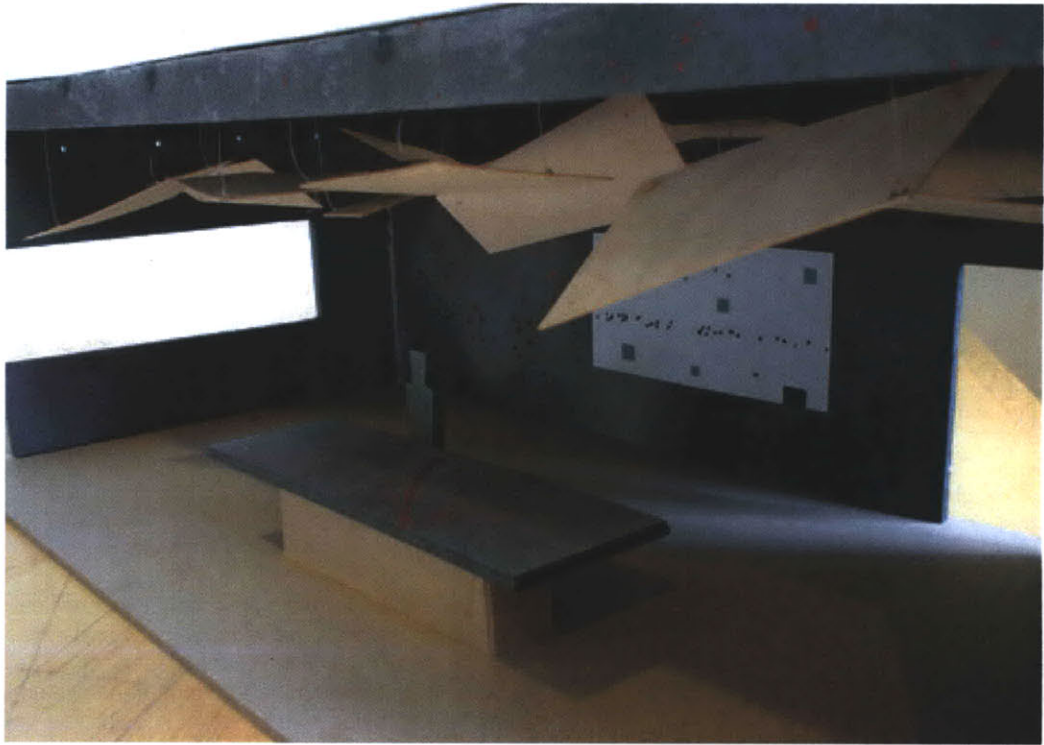
## PHYSICAL MODEL OF AGGREGATE SYSTEM

This model is a demonstration of the computational units applied in aggregate form as a architectural surface. The units can be linked to outside sensors depending on the nature of the program. The frame will contain a skylight when possible, but this is not essential. In this model, the system involves ambient sensing, but the panels are not linked to skylights.

**TRANSLATION:** The panels can drop to create zones of intimate space, or they can rise to create more open space.

**ROTATION:** The panels rotate to change their optical parameters. Above each panel is a set of lights, usually a skylight and an interior light. The rotation of the panel in conjunction with the time of day and type of illumination give the space ranges of dark and light.

**INPUT:** This collection of panels respond to direct user input in the form of an array of switches and also respond to ambient sensors detecting levels of visible and infrared light within the space. In this way the surface will respond to momentary shifts of light such as passing clouds as well as cyclical changes occurring daily, seasonally, or even longer. Infrared sensing allows the system to see where people are inhabiting the space and to respond accordingly.



## COMPUTATIONAL MODEL OF AGGREGATE SYSTEM

### ADAPTIVE SURFACE v 2.0

<http://web.mit.edu/jroth/www/thesis/>

This model offers an interactive look at how the system will respond once implemented. It contains a three dimensional study of the architectural form as well as a way to prototype the embedded computation.

The surface drew on the growth and decay model of the Adaptive Grid no. 001 applet, and was programmed in the proce55ing suite for Java, developed by Ben Fry and Casey Raes of the MIT Media Lab. The panels initially respond quite dramatically to user input or changes in ambient sensors, but the system then settles to a more stationary state. This base state is constantly updating and represents the internal history of the aggregate system.

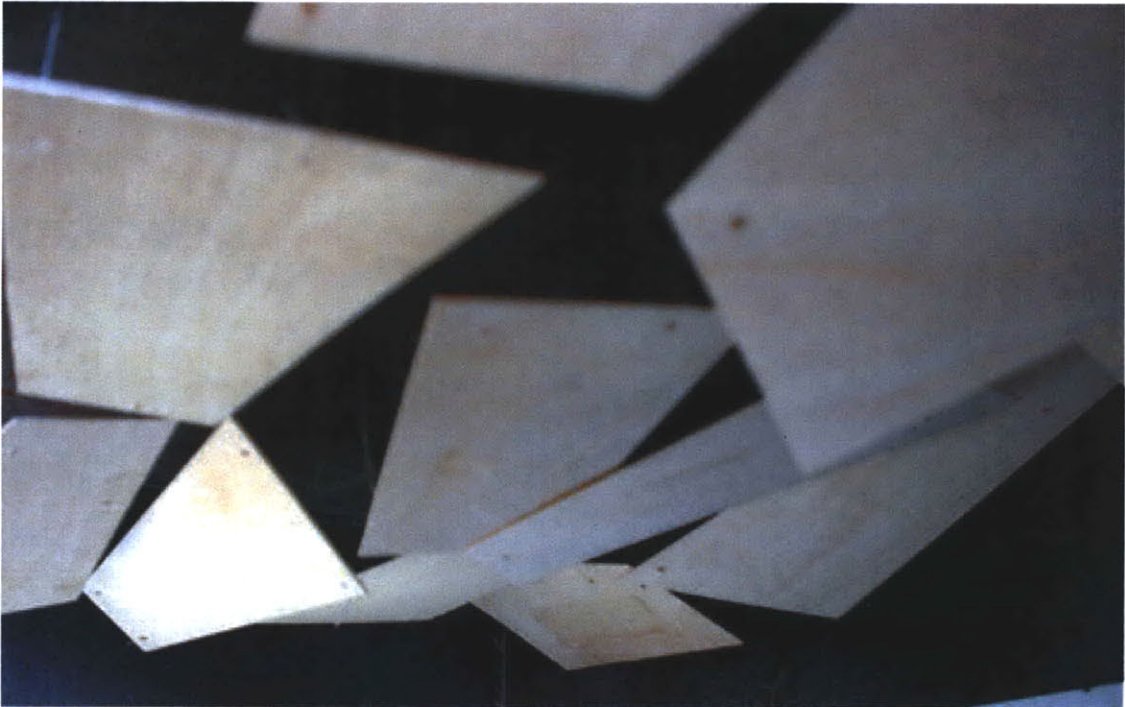
The translation from computational to physical space requires the surface to obey physical laws. Rotation and translation of the panels must be limited physically, and the freedom of the system must be contained to a range of possibilities. In previous cases of computational architecture being realized as physical form, there is often a gap between the envisioned computational model and the resulting physical model. This system attempts to model a real situation of aggregate panels. For this reason, the freedom and complexity of the evolutionary algorithm is replaced by a more practical adaptive algorithm until the physical implementation reaches the point where it can handle such complexity.







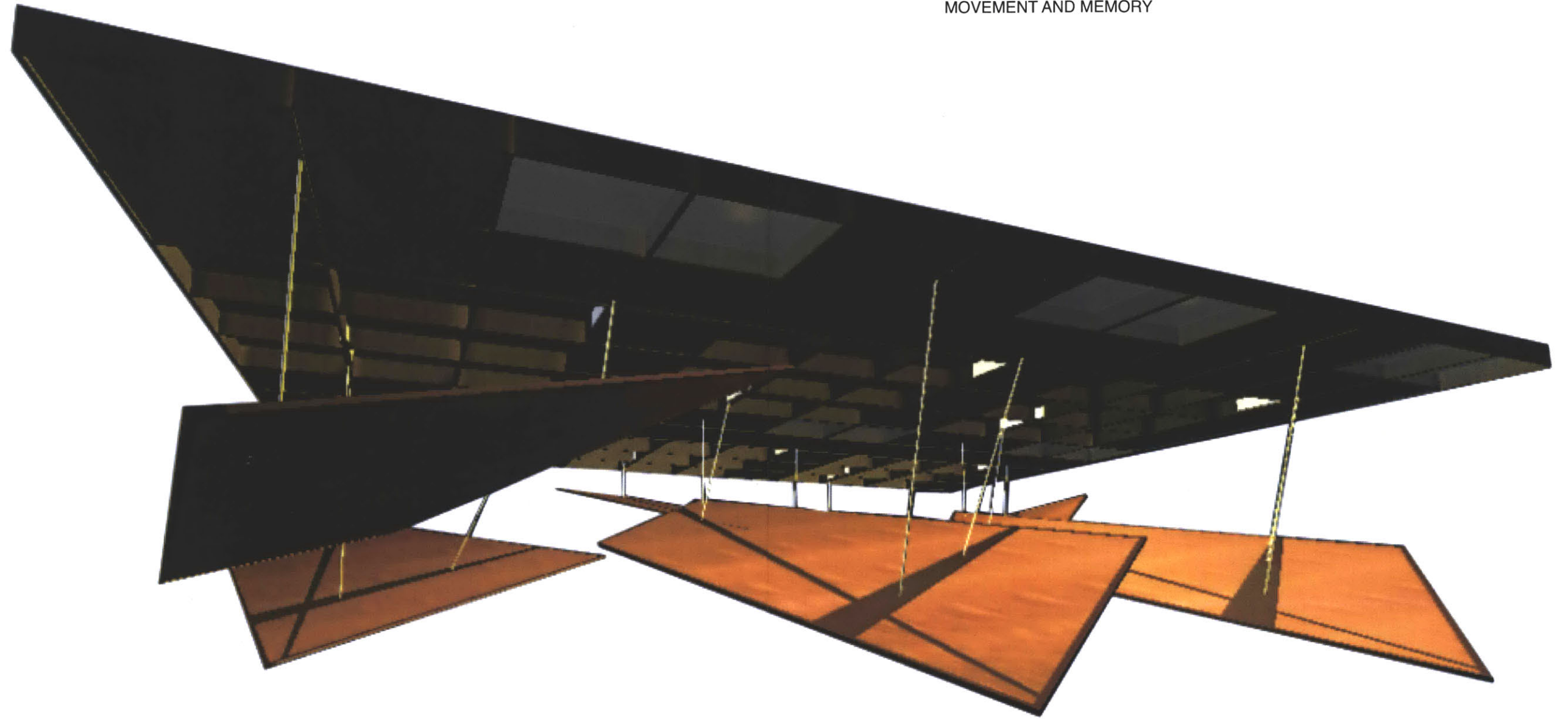
VIEWS OF COMPUTATIONAL AND PHYSICAL MODEL





## ADAPTIVE SURFACE IN AGGREGATE FORM

ADAPTIVE SURFACE IN AGGREGATE FORM  
MOVEMENT AND MEMORY



## V. CONCLUSION





## ISSUES OF DENSITY AND INTENSITY

This thesis is concerned with the forms of architectural change over time. Differentiation over time can be emphasized by the variations between a static repeated object, or by the variations between an object and itself in the form of a kinetic set of elements that change relative to each other. In the first case, it is the movement of the user through a sequence of repetition and variation that establishes the durations of the system. In the second, it is the physical space itself that fundamentally shifts over time, exhibiting duration.

Shifting densities and intensities can produce surprising architectural results. Recursion is a computational concept where an algorithm calls itself to solve a problem. Recursion deals with the cumulative variation that develops when a sequence is altered slightly and repeated. A history develops and informs the next iteration of the algorithm. Each state of the system is part of a potentially infinite set of states. This idea can be applied to architecture to produce spatial results that develop a cumulative complexity. Most importantly, the results can be in the form of growth or decay -- the system does not have to be directed at a specific end or solution.

My work is a proposal for an architecture of densities that shift over time according to rules of growth and decay. In this way the physical form is able to develop an internal history that responds and reacts to the passage of time. When built objects determine their own rates of change, they begin to exhibit the complexity of the unbuilt world. Such an architecture of intensities emphasizes that the greatest difference between the artificial and natural worlds might simply be one of duration.



## VI. ADDITIONAL MATERIALS



## WORKS CITED

DE LANDA, MANUEL. ONE THOUSAND YEARS OF NONLINEAR HISTORY. CAMBRIDGE: ZONE BOOKS, 2000.

DE LANDA, MANUEL. "NONORGANIC LIFE" ZONE 6. EDS. JONATHAN CURRY AND SANFORD KWINTER. CAMBRIDGE: ZONE BOOKS, 1992.

DELEUZE, GILLES. BERGSONISM. CAMBRIDGE: ZONE BOOKS, 1990.

DELEUZE, GILLES. "MEDIATORS" " ZONE 6. EDS. JONATHAN CURRY AND SANFORD KWINTER. CAMBRIDGE: ZONE BOOKS, 1992.



## SOURCE CODE FOR EVOLUTIONARY APPLLET no. 002

```
/*
    evo6.java evolution with reproduction, crossover, mutation
*/

import java.awt.*;
import java.applet.Applet;
import java.util.Date;

import java.awt.event.*;
import java.awt.geom.*;
import java.lang.Math.*;
import java.lang.StringBuffer;

import Randomizer;
import evoOrganism;
import simpleOrg;

public class evo6 extends Applet
    implements MouseListener, MouseMotionListener, Runnable {

    public static evo6 instance;

    //for mouse dragging:
    boolean moving = false;
    int mousedown;
    int mouseup;

    //for double buffering:
    Dimension size;
    Image buffer;
    Graphics bufferGraphics;
    Thread animator;
    boolean please_stop;

    //randomizer with time-based seed:
    Date time = new Date();
    long seed = time.getTime();
    Randomizer r = new Randomizer(seed);

    //global colors:
    int co = 0;

    Color c3 = new Color(235, 240, 230);
    Color c4 = new Color(168, 168, 139);

    Color grey = new Color(196, 196, 196);
    Color white = new Color(255, 255, 255);
    Color lime = new Color(209, 245, 99);

    Color block = new Color(117, 117, 95);
    Color band = new Color(246, 246, 204);
    Color bound = new Color(101, 101, 75);

    //pause for runtime:
    int pause_factor = 15;
    int special_clock = 0;
```

```

//general framework for evolution:
int number_living = 0;
int number_simple = 0;

//specific variables for applet:
evoOrganism[] burp = new evoOrganism[300];
simpleOrg[] sim = new simpleOrg[300];
int temp_x, temp_y, temp_size, temp_mob, temp_col, temp_amp;
int temp_width, temp_height;
int initial_blocks = 12;
//int initial_barriers = 3;
//int initial_attract = 2;

int bX, bY, bD, bW, bH, bA, bM, bL, bS;

dataSpace data = new dataSpace();

//testing and expanding barriers:
boolean open_space;
boolean expanding;
int which_simple;

//selection of mates:
int temp_sexy = 0;
int possible_mates = 0;
int sexy_one = 0;
int sexy_two = 0;
boolean mating = false;
int[] best = new int[2];
int[] second_best = new int[2];
int male;
int female;
int danger_x, danger_y, dense_x, dense_y;

//genetic reproduction:
int number_genes = 6;
int[] male_genes = new int[6];
int[] female_genes = new int[6];
int[] genes_one = new int[6];
int[] genes_two = new int[6];

//life_cycle:
int fill_here = 0;

//occupancy rules:
int regions = data.regions;
int width = data.grid_width;
int height = data.grid_height;

//init method is similar to the main method in a java application

public void init() {
    instance = this;
    size = this.size();
    buffer = this.createImage(size.width, size.height);
    bufferGraphics = buffer.getGraphics();

    addMouseListener(this);
    addMouseMotionListener(this);

    //dataSpace data = new dataSpace();

```



```

        //start with 10 blocks on their own:
        for (int i=0; i<initial_blocks; i++) {
            temp_x = r.randomInt(500)+50;
            temp_y = r.randomInt(240)+30;
            temp_size = r.randomInt(40)+10;
            temp_mob = r.randomInt(6)+5;
            temp_col = r.randomInt(31);
            temp_amp = r.randomInt(4)+2;

            burp[i] = new evoOrganism(temp_x, temp_y, temp_size, temp_mob, temp_col, temp_amp);
            data.addOrg(burp[i]);
            number_living += 1;
        }
        //System.out.println("initialized" + number_living + "organisms");
    }

    public void paint(Graphics g) {

        //draw background
        this.setBackground(c4);
        bufferGraphics.setColor(this.getBackground());
        bufferGraphics.fillRect(0, 0, size.width, size.height);

        //draw grid
        bufferGraphics.setColor(white);
        for(int gX=0; gX<data.grid_width; gX++) {
            int wave = r.randomInt(3);
            int grid_x = data.grid_x[gX];
            //bufferGraphics.fillRect(grid_x, 0, wave, 300);
            bufferGraphics.drawLine(grid_x, 0, grid_x, 300);
        }
        for(int gY=0; gY<data.grid_height; gY++) {
            int wave2 = r.randomInt(3);
            int grid_y = data.grid_y[gY];
            //bufferGraphics.fillRect(0, grid_y, 600, wave2);
            bufferGraphics.drawLine(0, grid_y, 600, grid_y);
        }

        //draw barriers
        bufferGraphics.setColor(lime);
        for (int j=0; j<number_simple; j++) {
            bX = sim[j].getX();
            bY = sim[j].getY();
            bW = sim[j].getSize();

            bufferGraphics.fillRect(bX, bY, bW, bW);
        }
        if(expanding) {
            sim[which_simple].increaseSize();
            //try { Thread.sleep(5); } catch (InterruptedException e) { ; }
        }

        //draw organisms
        for (int i=0; i<number_living; i++) {
            co = burp[i].getColor();
            bX = burp[i].getX();
            bY = burp[i].getY();
            bD = burp[i].getSize();
            int draw_size = bD;
            bA = burp[i].fixedAmp();
            bM = burp[i].getMob();
            bS = burp[i].getSexy();
            bL = burp[i].getLife();
        }
    }
}

```

```

        int jump = burp[i].getAmp()/2;

        bufferGraphics.setColor(block);
        bufferGraphics.fillRect(bX, bY, draw_size-1, draw_size-1);

        bufferGraphics.setColor(band);
        int band_height = draw_size/4;
        //a band for all of it's characteristics:
        bufferGraphics.fillRect(bX, bY, bA*10, band_height);
        bufferGraphics.fillRect(bX+bM, bY+band_height, draw_size-bM, band_height);
        bufferGraphics.fillRect(bX, bY+(2*band_height), bS/bD, band_height);
        bufferGraphics.fillRect(bX+co*2, bY+(3*band_height), draw_size-co*2, band_height+1);

        bufferGraphics.setColor(bound);
        bufferGraphics.drawRect(bX-1*bD+jump, bY-3*bD+jump, 9*bD, 5*bD);
    }

    //draw info buffer:
    bufferGraphics.setColor(white);
    bufferGraphics.fillRect(0, 299, 600, 340);

    bufferGraphics.setColor(lime);
    bufferGraphics.fillRect(0, 290, special_clock/10, 4);
    //put an overwrapped bar of timeline here.

    String living = new String("// " +String.valueOf(number_living) + " system organisms");
    bufferGraphics.setColor(bound);
    bufferGraphics.drawString(living, 10, 330);

    String inputs = new String("// " +String.valueOf(number_simple) + " user inputs");
    //bufferGraphics.setColor(lime);
    bufferGraphics.drawString(inputs, 10, 318);

    g.drawImage(buffer, 0, 0, this);
}

public void update(Graphics g) { paint(g); }

    //method for runnable:
    public void run() {
        while(!please_stop) {

            if (special_clock%10 == 0) {
                this.moveSims();
                this.killSims();
            }

            if (special_clock%50 == 0) {
                this.calculateOccupants();
                this.gridResponse();
            }

            if (special_clock%250 == 0) {
                this.selectMates();
                this.reproduce();
                this.selectKills();
                this.kill();
            }

            special_clock++;
            //change to 10(?) before posting to web
            try { Thread.sleep(12); } catch (InterruptedException e) { ; }
            repaint();
        }
    }

```

```

    }
    animator = null;
}

public void calculateOccupants() {
    int org_x;
    int org_y;
    int left;
    int right;
    int top;
    int bottom;

    //zero the number of occupants in each region:
    data.zeroOccupants();

    //check each organism and assign to the dataSpace:
    for (int i=0; i<number_living; i++) {
        org_x = burp[i].getX();
        org_y = burp[i].getY();
        for (int w=0; w<(width-1); w++) {
            left = data.grid_x[w];
            right = data.grid_x[w+1];
            //keep the on the edge vague - let organisms exploit it
            if ((left < org_x) && (right > org_x)) {
                for(int h=0; h<(height-1); h++) {
                    top = data.grid_y[h];
                    bottom = data.grid_y[h+1];
                    if ((top < org_y) && (bottom > org_y)) {
                        data.gridAdd(w, h, i);
                    }
                }
            }
        }
    }
}

public void gridResponse() {
    int occ;
    for(int i=0; i<(data.grid_width-1); i++) {
        for(int j=0; j<(data.grid_height-1); j++) {
            occ = data.occupants[i][j][0];

            //region of 2 blocks
            if (occ == 2) {
                int last_in = data.grid_org[i][j][occ-1];

                int mobile = burp[last_in].getMob();
                int x_move = (r.randomInt(2000)%mobile)*2 - mobile/2;
                int y_move = (r.randomInt(2000)%mobile)*2 - mobile/2;
                //makes a random move
                burp[last_in].move(x_move, y_move);

                // gets a color change - more life ability
                int color_change = r.randomInt(2000)%4 -2;
                burp[last_in].changeColor(1);
            }

            //region of more
            if (occ > 2) {
                int last_in = data.grid_org[i][j][occ-1];

                while(occ>3) {
                    burp[last_in].getKilled();
                    occ = data.occupants[i][j][0] - 1;
                }
            }
        }
    }
}

```

```

        last_in = data.grid_org[i][j][occ-1];
    }
    //now last_in should be 3:
    int mobile = burp[last_in].getMob();
    int x_move = (r.randomInt(2000)%mobile)*4 - mobile/4;
    int y_move = (r.randomInt(2000)%mobile)*4 - mobile/4;

    burp[last_in].move(x_move, y_move);
}
}
}

public void killSims() {
    if (number_simple == 1) {
        if (sim[0].life < 1) number_simple = 0;
    }
    else for (int i=0; i<number_simple-1; i++) {
        if (sim[i].life < 1) {
            sim[i] = sim[i+1];
            number_simple -= 1;
        }
    }
}

int line_x;
int line_y;
public void moveSims() {
    for (int i=0; i<number_simple; i++) {
        temp_x = sim[i].getX();
        temp_y = sim[i].getY();

        //check to see if the x-point lies on a grid line:
        for (int j=0; j<width; j++) {
            line_x = data.grid_x[j];
            //hitting a verticle:
            if (Math.abs(line_x-temp_x) < 2) {
                if(sim[i].left data.grid_x[j] -= sim[i].getSize());
                else data.grid_x[j] += sim[i].getSize();
                sim[i].left =! sim[i].left;
            }
        }

        for (int k=0; k<height; k++) {
            line_y = data.grid_y[k];
            if (Math.abs(line_y-temp_y) < 2) {
                if(sim[i].down data.grid_y[k] += sim[i].getSize());
                else data.grid_y[k] -= sim[i].getSize();
                //System.out.println("vert");
                sim[i].down =! sim[i].down;
            }
        }

        sim[i].move();
    }
}

//tally sexiness and 2 most sexy that aren't stranded will mate:
public void selectMates() {

    best[0] = 0;           //highest sexy value
    best[1] = 0;           //corresponding org index
    second_best[0] = 0;
}

```

```

second_best[0] = 0;
//possible_mates = 0;

int danger_x = data.simple_X();
int danger_y = data.simple_Y();
int dense_x = data.average_X();
int dense_y = data.average_Y();

if(number_living>1) {
    this.selectOne();
    this.selectTwo();
    male = best[1];
    female = second_best[1];
    mating = true;
    //System.out.println("sexiest" + best[1] + "mating" + second_best[1]);
}
}

public void selectOne() {

    //checking distance from danger
    //including color as an attractive quality

    for (int i=0; i<number_living; i++) {
        temp_sexy = 60;
        temp_x = burp[i].getX();
        if (number_simple != 1) {
            temp_sexy = Math.abs(temp_x - danger_x);
            temp_sexy += Math.abs(temp_y - danger_y);
        }
        temp_sexy -= (burp[i].getSize());
        temp_sexy += (burp[i].getColor());
        burp[i].setSexy(temp_sexy);
        //System.out.println("org" + i + "barrier/size" + temp_sexy);
    }

    //selecting the most fit:
    for (int j=0; j<number_living-1; j++) {
        int sexy_current = burp[j].getSexy();
        if (sexy_current > best[0]) {
            best[0] = sexy_current;
            best[1] = j;
            burp[j].selected = true;
        }
    }
}

public void selectTwo() {

    //checking distance from others
    //including Age as an attractive quality
    for (int i=0; i<number_living; i++) {
        temp_x = burp[i].getX();
        temp_sexy = Math.abs(temp_x - dense_x);
        temp_sexy -= Math.abs(temp_y - dense_y);
        temp_sexy += (burp[i].getAge());
        burp[i].setSexy(temp_sexy);
        //System.out.println("org" + i + "diversity/color" + temp_sexy);
    }

    //selecting the most fit:
    //ignore the already selected
    for (int j=0; j<number_living-1; j++) {

```

```

        if (!burp[j].selected) {
            int sexy_current = burp[j].getSexy();
            if (sexy_current > second_best[0]) {
                second_best[0] = sexy_current;
                second_best[1] = j;
            }
        }
    }
}

public void reproduce() {
    if(mating) {

        int crosspoint = r.randomInt(2000)%6;
        int one_mutation = r.randomInt(2000)%6;
        int two_mutation = r.randomInt(2000)%6;

        male_genes[0] = burp[male].getX();
        male_genes[1] = burp[male].getY();
        male_genes[2] = burp[male].getSize();
        male_genes[3] = burp[male].getMob();
        male_genes[4] = burp[male].getColor();
        male_genes[5] = burp[male].getAmp();

        female_genes[0] = burp[female].getX();
        female_genes[1] = burp[female].getY();
        female_genes[2] = burp[female].getSize();
        female_genes[3] = burp[female].getMob();
        female_genes[4] = burp[female].getColor();
        female_genes[5] = burp[female].getAmp();

        //crossover
        for (int i=0; i<crosspoint; i++) {
            genes_one[i] = male_genes[i];
            genes_two[i] = female_genes[i];
        }
        for (int j=crosspoint; j<6; j++) {
            genes_one[j] = female_genes[j];
            genes_two[j] = male_genes[j];
        }

        //mutation

        genes_one[one_mutation] += (r.randomInt(2000)%12 - 5);
        //System.out.println("mutating gene" + one_mutation);
        genes_two[two_mutation] += (r.randomInt(2000)%12 - 5);

        burp[number_living] = new evoOrganism(genes_one[0], genes_one[1], genes_one[2], genes_one[3],
genes_one[4], genes_one[5]);
        data.addOrg(burp[number_living]);
        number_living += 1;
        burp[number_living] = new evoOrganism(genes_two[0], genes_two[1], genes_two[2], genes_two[3],
genes_two[4], genes_two[5]);
        data.addOrg(burp[number_living]);
        number_living += 1;

        mating = false;

    }
}

public void selectKills() {
    // give everyone a year of age, and calculate their life_ability
    for (int i=0; i<number_living; i++) {

```

```

        burp[i].addAge();
        burp[i].tallyLife();
    }
}

public void kill() {
    //System.out.println("alive" + number_living);
    fill_here = 0;
    for(int i=0; i<(number_living); i++) {
        if(burp[i].getAlive()) {
            burp[fill_here] = burp[i];
            fill_here += 1;
        }
    }
    number_living = fill_here;
    //System.out.println("alive" + number_living);
}

public void start() {
    if (animator == null) {
        please_stop = false;
        animator = new Thread(this);
        animator.start();
    }
}

public void stop() { please_stop = true; }

// will he notice this - yes

//methods for mouse press, clicked, entered, exited -- MouseListener
public void mousePressed(MouseEvent event) {
    open_space = true;
    temp_x = event.getX();
    temp_y = event.getY();
    if (number_simple == 0) {
        temp_size = 8;
        temp_x = temp_x - temp_width/2;
        temp_y = temp_y - temp_width/2;

        sim[number_simple] = new simpleOrg(temp_x, temp_y, temp_size);
        data.addSim(sim[number_simple]);
        number_simple += 1;
        open_space = false;
    }
    //check to see if you are already on a void space - make new or expand existing.
    else {
        for (int i=0; i<number_simple; i++) {
            int block_x = sim[i].getX();
            int block_y = sim[i].getY();
            int size = sim[i].getSize();

            if( temp_x>block_x && temp_x<(block_x+size) ) {
                if( temp_y>block_y && temp_y<(block_y+size) ) {
                    //bar[i].increaseSize();
                    which_simple = i;
                    expanding = true;
                    //try { Thread.sleep(5); } catch (InterruptedException e) { ; }
                    open_space = false;
                }
            }
        }
    }
}

```

```

    }
    if(open_space) {
        temp_size = 8;
        temp_x = temp_x - temp_size/2;
        temp_y = temp_y - temp_size/2;

        sim[number_simple] = new simpleOrg(temp_x, temp_y, temp_size);
        data.addSim(sim[number_simple]);
        number_simple += 1;
    }
}

public void mouseReleased(MouseEvent event) {
    expanding = false;
}

public void mouseClicked(MouseEvent event) {}
public void mouseEntered(MouseEvent event) {}
public void mouseExited(MouseEvent event) {}

//methods for mouse moved, dragged -- MouseMotionListener
public void mouseMoved(MouseEvent event) {}
public void mouseDragged(MouseEvent event) {
    //if(moving)->do something
}
}
}

```



```

/*
    evoOrganism:
        characteristics: position, size, age, color, mobility, distance from danger, proximity to food
*/

import java.awt.*;
//import java.lang.Math.*;
//import java.util.Date;
import Randomizer;
import dataSpace;

public class evoOrganism {

    //genetic code: private values
    int x_val;
    int y_val;
    int size;
    int mobility;
    int color_val;
    int amplitude;

    //life ability fields:
    int life_ability;
    int sexy;
    boolean alive;
    int age;
    int distance_bad;
    int distance_good;
    public int diversity_bonus = 0;
    int initial_life = 20;

    //graphic characteristics:
    public boolean anchored;
    public int anchor_clock;
    boolean vibrate;
    public boolean stranded;
    public boolean selected;

    Randomizer r = new Randomizer(2);
    //public netPoint netpt;

    //object should be initialized with all 6 elements of it's genetic code:
    public evoOrganism(int x, int y, int delta, int mobile, int col, int amp) {

        //genetic code - cannot be altered except in reproduction/mutation sequence.
        x_val = x;
        y_val = y;
        size = delta;
        mobility = mobile;
        color_val = col;
        //keep color in range!
        if(color_val > 31) color_val = 31;
        amplitude = amp;

        //state characteristics
        alive = true;
        anchored = false;
        vibrate = true;
        stranded = false;
        age = 0;
    }
}

```

```

        //determining factors in
        life_ability = initial_life;
        sexy = 0;
    }

    public int getX() { return x_val; }
    public int getY() { return y_val; }
    public void setX(int x) {x_val = x;}
    public void setY(int y) {y_val = y;}
    public void move(int x, int y) {
        x_val += x;
        y_val += y;
    }

    public int getAmp() {return r.randomInt(amplitude);}
    public int fixedAmp() {return amplitude;}
    public void setAmp(int amp) {amplitude = amp;}

    public int getSize() {return size;}
    public void setSize(int d) {
        int delta_size = d - size;
        x_val -= delta_size;
        y_val -= delta_size;
        size = d;
    }
    public void increaseSize(int d) {
        x_val -= d/2;
        y_val -= d/2;
        size += d;
    }

    public int getColor() {return color_val;}
    public void setColor(int c) {
        color_val = c;
        if(color_val>31) color_val = 31;
    }
    public void changeColor(int a) {
        color_val += a;
        if(color_val>31) color_val = 31;
        if(color_val<0) color_val = 0;
    }

    public int getMob() {return mobility;}
    public void setMob(int m) {mobility = m;}

    public void addAge() {age += 1;}
    public int getAge() {return age;}

    public void expand(int how_much) {
        if(!anchored) {
            for(int i=0; i<how_much; i++) {
                this.increaseSize(1);
            }
        }
    }

    public int getSexy() {return sexy;}
    public void setSexy(int s) {sexy = s;}

    public void tallyLife() {
        int reaper = r.randomInt(2000)%15;

```

```
life_ability = initial_life;
life_ability -= age/3;
//color will also play a role in action/response
life_ability += 5-color_val/8;
if(stranded) life_ability -= 1;
if(size>50) life_ability -= 1;
if(amplitude>12) life_ability -= 1;
if(mobility<12) life_ability -= 1;

if(reaper>life_ability) alive = false;

if (x_val > 500) alive = false;
if (x_val < -10) alive = false;
if (y_val > 300) alive = false;
if (y_val < -10) alive = false;
}

public void getKilled() {alive = false;}

public boolean getAlive() {return alive;}
public int getLife() {return life_ability;}

}
```



```

/*
    evoBarrier.java:

    characteristics: position, size, severity
*/

import java.awt.*;
import Randomizer;
import java.util.Date;

public class simpleOrg {

    //state: private values
    int x_val;
    int y_val;
    int delta;

    //life span:
    public int life;

    //movement cues:
    public int craze;
    public boolean left;
    public boolean down;

    Date time = new Date();
    long seed = time.getTime();
    Randomizer r = new Randomizer(seed);

    //object should be initialized with all position and size
    public simpleOrg(int x, int y, int d) {

        x_val = x;
        y_val = y;
        delta = d;

        craze = r.randomInt(99)%3+1;
        life = 30*craze;

        //state characteristics
        int mover = r.randomInt(99)%2;
        if (mover == 1) left = true;
        else left = false;
        mover = r.randomInt(99)%2;
        if (mover == 1) down = true;
        else down = false;

    }

    public int getX() { return x_val; }
    public int getY() { return y_val; }
    public int getSize() {return delta; }

    public void increaseSize() {
        x_val -= 1;           //d/2;
        y_val -= 1;           //d/2;
        delta += 2;
    }

    public void move() {

```

```
        if (left) x_val -= craze;
        else x_val += craze;
        if (down) y_val += craze;
        else y_val -= craze;

        life -= 1;
    }

    public void expand(int how_much) {
        for(int i=0; i<how_much; i++) {
            this.increaseSize();
            evo6.instance.repaint();
        }
    }
}
```

```

/*
    dataSpace.java
    tracking positions and densities of the world
*/

import java.awt.*;
import evoOrganism;
import simpleOrg;

public class dataSpace {

public static dataSpace data_instance;

    int number_orgs;
    int number_sim;
    int x_ave;
    int y_ave;
    int simple_x;
    int simple_y;

    int org_xvals[] = new int [1000];
    int org_yvals[] = new int [1000];
    int sim_xvals[] = new int[1000];
    int sim_yvals[] = new int[1000];

    public static int grid_width = 7;
    public static int grid_height = 4;

    public int regions = (grid_width-1)*(grid_height-1);
    //public int[] occupants = new int[regions];
    //public int[][] occupants = new int[(grid_width-1)][(grid_height-1)];

    public int grid_x[] = new int[grid_width];
    public int grid_y[] = new int[grid_height];

    //oversized for convenience:
    public int occupants[][][] = new int [grid_width][grid_height][1];
    public int grid_org[][][] = new int [grid_width][grid_height][60];

    public dataSpace() {
        number_orgs = 0;
        number_sim = 0;
        x_ave = 0;
        y_ave = 0;
        simple_x = 0;
        simple_y = 0;
        for (int i=0; i<grid_width; i++) {
            //System.out.println("here");
            grid_x[i] = 100*(i)-1;
        }
        for (int j=0; j<grid_height; j++) {
            grid_y[j] = 100*(j)-1;
        }
        for (int k=0; k<grid_width; k++) {
            occupants[k][0][0] = 0;
            occupants[k][1][0] = 0;
            occupants[k][2][0] = 0;
        }
    }

    public void zeroOccupants() {

```

```

        for (int k=0; k<grid_width; k++) {
            occupants[k][0][0] = 0;
            occupants[k][1][0] = 0;
            occupants[k][2][0] = 0;
        }
    }

    //to move a block: add it to new grid, and then subtract one from old grid.
    public void gridAdd(int width, int height, int which_one) {
        int placement = occupants[width][height][0];
        occupants[width][height][0] += 1;
        grid_org[width][height][placement] = which_one;
    }

    public void gridSub(int width, int height) {
        //the last one in must get the fuck out first;
        //other functions can't get at it once we shift the index.
        occupants[width][height][0] -= 1;
        //most likely have to do a major shift ...
    }

    public void addOrg(evoOrganism org) {
        org_xvals[number_orgs] = org.getX();
        org_yvals[number_orgs] = org.getY();
        number_orgs ++;
        x_ave = ((number_orgs-1)*x_ave + org.getX())/number_orgs;
        y_ave = ((number_orgs-1)*y_ave + org.getY())/number_orgs;
        //System.out.println("averageX =" + x_ave +", averageY =" + y_ave);
    }

    //this needs to be double checked!
    public void removeOrg(int org_number) {
        for (int i=org_number; i<number_orgs-1; i++) {
            org_xvals[i] = org_xvals[i+1];
            org_yvals[i] = org_yvals[i+1];
            //System.out.println("shift");
        }
        number_orgs -= 1;
        x_ave = 0;
        y_ave = 0;
        for (int j=1; j<number_orgs; j++) {
            x_ave = ((j-1)*x_ave + org_xvals[j]) / j;
            y_ave = ((j-1)*y_ave + org_yvals[j]) / j;
        }
        //System.out.println("averageX =" + x_ave +", averageY =" + y_ave);
    }

    public int average_X() { return x_ave; }
    public int average_Y() { return y_ave; }

    public void addSim(simpleOrg sim) {
        sim_xvals[number_sim] = sim.getX();
        sim_yvals[number_sim] = sim.getY();
        number_sim ++;
        simple_x = ((number_sim-1)*simple_x + sim.getX())/number_sim;
        simple_y = ((number_sim-1)*simple_y + sim.getY())/number_sim;
        //System.out.println("simple_averageX =" + simple_x +", averageY =" + simple_y);
    }

    public int simple_X() {return simple_x;}
    public int simple_Y() {return simple_y;}

```



```
public void removeSim(int sim_number) {
    for (int i=sim_number; i<number_sim-1; i++) {
        sim_xvals[i] = sim_xvals[i+1];
        sim_yvals[i] = sim_yvals[i+1];
        //System.out.println("shift");
    }
    number_sim -= 1;
    simple_x = 0;
    simple_y = 0;
    for (int j=1; j<number_sim; j++) {
        simple_x = ((j-1)*x_ave + org_xvals[j]) / j;
        simple_y = ((j-1)*y_ave + org_yvals[j]) / j;
    }
    //System.out.println("simpleX =" + simple_x +", simpleY =" + simple_y);
}
}
```



