

Spatial Computing

Simon Greenwold

B.S., English & Applied Math
Yale University, June 1995

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning, in partial
fulfillment of the requirements for the degree of
Master of Science in Media Arts and Sciences
at the
Massachusetts Institute of Technology
June 2003

© Massachusetts Institute of Technology
All rights reserved

Author: Simon Greenwold
Program in Media Arts and Sciences
May 16, 2003

Certified by: John Maeda
Associate Professor of Media Arts and Sciences
Thesis Supervisor

Accepted by: Dr. Andrew B. Lippman
Chair, Departmental Committee on Graduate Studies
Program in Media Arts and Sciences

Spatial Computing

Simon Greenwold

Submitted to the Program in Media Arts and Sciences, School of Architecture and Planning, on May 16, 2003, in partial fulfillment of the requirements for the degree of Master of Science in Media Arts and Sciences

Abstract

Spatial computing is human interaction with a machine in which the machine retains and manipulates referents to real objects and spaces. It is an essential component for making our machines fuller partners in our work and play. This thesis presents a series of experiments in the discipline and analysis of its fundamental properties.

Spatial Computing

Simon Greenwold



thesis reader



thesis reader



thesis advisor

Joseph A. Paradiso

Associate Professor

Co-Director, Things That Think Consortium

MIT Media Laboratory

Chris Csikszentmihályi

Assistant Professor of Media Arts and Sciences

MIT Program in Media Arts and Sciences

John Maeda

Associate Professor

MIT Program in Media Arts and Sciences

Acknowledgements

A lot of people helped me write this.

Ben expounded,
Tom explained,
Megan offered,
James thought,
Justin twisted,
Elizabeth read,
Michael invented,
Missy laughed,
Ollie wondered,
Stephanie reminded,
Axel agreed,
Chris connected,
Joe knew,
John reacted,
Jeanna fed,
Craig prodded,
Tim extrapolated,
Jeana worried,
Ryan subverted,
Michelle gave,
Tad questioned,
Eric responded,
Jake reflected,
Thea felt,
Jess walked,
Sharon painted,
Joanne lavished,
Richard played,
Rebecca conspired,
Jeremy argued,
Mom checked,
Dad worked,
Diana played,
Amanda asked,
Roger understood,
Charlotte smiled,
Jenny did a million things every day.

Contents

Part I: Framework

1. Preliminaries.....	8
1.1 Introduction	8
1.2 Definition	11
2. Background	13
2.1 History	13
2.1.1 The Machine in Space	13
2.1.2 Space in the Machine	16
2.1.3 Networked Space	20
2.1.4 The Denial of Space	21
2.3 The Problems With Realism	23
2.4 The Problems With Interactivity	30
3. Motivation.....	32
4. Enter Spatial Computing	33
5. Methodology.....	34
6. Precedents	35
7. Roadmap of Experiments	40
7.1 Installation	40
7.2 Internaut	41
7.3 Stomping Ground	41
7.4 Hotpants/LittleVision	42
7.5 Pointable Computing	43
7.6 EyeBox	44
8. Installation.....	47
8.1 Introduction	47
8.2 System Description	48
8.3 Technical Details	51
8.4 Precedents	52
8.5 Evaluation and Critique	54
8.6 Future Work	60

Part II: Experiments

9. Internaut.....	62
9.1 Introduction	62
9.2 Technical Description	63
9.3 Precedents	65
9.4 Evaluation and Critique	67
9.5 Future Work	70
10. Stomping Ground	71
10.1 Introduction	71
10.2 System Description	71
10.3 Precedents	72
10.4 Evaluation and Critique	72
11. Hotpants/LittleVision	74
11.1 Introduction	74
11.2 System Description	74
11.3 Technical Details	75
11.4 Precedents	77
11.5 Evaluation and Critique	78
12. Pointable Computing	80
12.1 Introduction	80
12.2 System Description	82
12.3 Precedents	83
12.4 Use Scenarios	85
12.5.1 Universal Remote	85
12.5.2 Active Tagging	85
12.5.3 Getting and Putting	85
12.5.4 Instant Wiring	86
12.5.5 Reactive Surfaces	86
12.5 Evaluation and Critique	87
13. EyeBox.....	90
13.1 Introduction	90
13.2 System Description	91
13.3 Motivation	92
13.4 Technical Description	92
13.5 Precedents	95
13.6 Design and Operation	97
13.7 Evaluation and Critique	100

Part III: Evaluation	
14. Summary Conclusions.....	102
14.1 Editing	102
14.1.1 Suggestion vs. realism	102
14.1.2 Literalness vs. metaphor	103
14.1.3 Design vs. engineering	104
14.2 Qualities of Interactions	104
14.2.1 Full vs. limited	105
14.2.2 Feedback vs. one-way control	105
14.2.3 Predictability vs. inconsistency	105
14.3 Shortcuts	106
14.3.1 Relativity of perception	106
14.3.2 Opacity	107
14.4 The Future of Spatial Computing	108
14.4.1 Technologies	108
14.4.2 Applications	109
15. References	111

Appendix A. Other Experiments**117**

Appendix B. SBalls code**120**

Appendix C. Hotpants tech. docs**129**

1.1 Introduction

We have arrived at a critical point in the history of the machine in space. Engineers are rapidly banishing the last moving parts from consumer electronics, allowing the machines to shrink into near invisibility. Bulky CRTs are yielding to flat panels, allowing us to embed them into the surfaces we use daily and to free up valuable “real estate” on our desks. The businesses of computer vision and graphics have pushed our abilities to recover spatial information from the world at large and represent it recognizably to the eye. The long-standing divide between the idealized spaces of computer science and the heavy, cluttered spaces of real-world engineering are wider than ever, polarizing research around the world. Now that computation’s denial of physicality has gone about as far as it can, it is time for a reclamation of space as a computational medium.

The sublimation of the embodied world has not gone unremarked or unchallenged. Criticism such as Katherine Hayles’ *How We Became Posthuman*, [Hayles, 1999] explores the intellectual and cultural roots of our sloughing the physical. In the laboratory, Rodney Brooks has shifted the focus of artificial intelligence research at MIT away from software systems to robots with “embodied intelligence,” theorizing that intelligence exists only in relation to a physical world [Brooks, 2001].

Recognizing the benefits of integrating systems into existing places and practices, research in virtual environments is growing to include facets of the real, albeit often heavily filtered and digitized. A recent survey of this trend, *Mixed Reality: Merging Real and Virtual Worlds*, organizes these efforts along an axis of “Extent of World Knowledge” [Milgram, 1999]. According to this heuristic, a computational system inside a purely real environment has no need to model the world at

all, whereas an entirely virtual environment must model the world in every particular. The systems organized on this scale and the principles derived from them properly belong to the field of Human-Computer Interaction (HCI), whose literature has ballooned beyond manageability.

In the middle of all this activity I place my own research agenda, *spatial computing*, which borrows from cultural criticism, theories of HCI, and systems of mixed reality, but belongs to none of them exclusively. This mongrel nature complicates the dialectic of its presentation. There is both a rhetorical and technical agenda to serve here. Therefore, in the interest of clarity, I organize my thesis into three parts.

Part I: Framework

I begin with a definition of spatial computing and then sketch the conditions that give it currency. Finally I give a brief chronology of the experiments I have undertaken in the field.

Part II: Experiments

This section fully presents the six completed projects. They are described roughly as they would be for technical publication. Each system had its own local set of concerns and conditions apart from the larger context of spatial computing. The analyses and critiques of each system in this section, is with regard to its particular objectives. An experiment (such as *Internaut*) that was not an unqualified success in achieving its own goals sometimes had a significant contribution to the broader study of spatial computing.

Part III: Analysis

This section presents a combination of technical and theoretical results. First I present an incomplete list of guiding principles for the

successful implementation of projects in spatial computing, synthesized from the technical analyses of each of the projects. Next I offer a guesses about the futures of the technologies involved in spatial computing. Finally I describe a sample application of the field.

But first we need a definition...



Figure 1.1. This project, *Installation*, allows users to place virtual objects in real space. It is a good example of spatial computing. (Discussed in detail below.)

1.2 Definition

I define spatial computing as human interaction with a machine in which the machine retains and manipulates referents to real objects and spaces. Ideally, these real objects and spaces have prior significance to the user. For instance, a system that allows users to create virtual forms and install them into the actual space surrounding them is spatial computing. A system that allows users to place objects from their environments into a machine for digitization is spatial computing. Spatial computing differs from related fields such as 3D modeling and digital design in that it requires the forms and spaces it deals with to pre-exist and have real-world valence. It is not enough that the screen be used to represent a virtual space—it must be meaningfully related to an actual place.

I use “virtual space” broadly here not just to refer to three-dimensional Cartesian worlds, but any space maintained by a computer and supposed to appeal to a human sense of space. By this definition a “desktop” in a graphical user interface is a virtual space. Similarly spatial computing does not necessarily take place in a three-dimensional representation. For many human purposes a piece of paper is better understood as a two-dimensional surface than a three-dimensional object. In fact, spatial computing may not present a space to the user at all. It necessarily maintains an internal representation of space, even if it is only implicit in collected data, but its interaction with a user need not be visual or spatial. The simplest example may be an auto-flushing toilet that senses the user’s movement away to trigger a flush. This is trivial spatial computing, but it qualifies. The space of the system’s engagement is a real human space.



Figure 1.2. The desktop is a virtual space. Notice here shading and occlusion.

The criterion that the objects and places in spatial computing have physical instantiation is not an arbitrary or trivial distinction. There are specific characteristics that make the production and analysis of spatial computing systems different from purely synthetic virtual systems. This distinction does not imply a value judgment—virtual systems have their place. However there are many cases, some discussed below, in which spatial computing could significantly benefit existing virtual systems.

It may seem that the category of computational systems that engage true space is too broad to tackle in a single thesis. That is likely true, and I wish to be careful with the generality of the claims I make. But I do not think that the diversity inside the topic defeats the purpose of considering it as a whole. Instead, I think it may be useful to do so in order to upset a traditional taxonomy, one which would not allow the analysis of physical systems next to software systems. In presenting spatial computing as an organizing principle, I allow several systems I have engineered to be brought into analysis together closely enough that they can shed light on one another.

2.1 History

Spatial computing proposes a tight linkage of the space in the machine (the space of digital representation) and the machine in space (physical presence). The starkness of the divide owes something to the fundamental differences of the media but a great deal also to social and historical construction. In order to understand our present dilemma it is necessary to examine a history of computation in physical space.

2.1.1 The Machine in Space

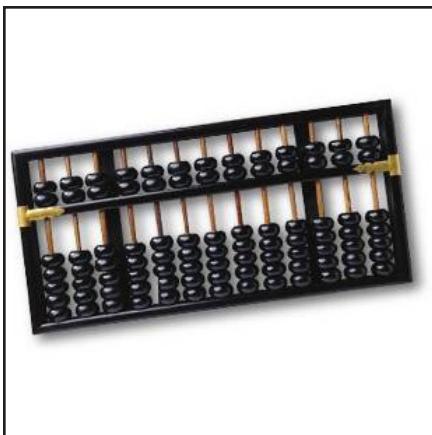


Figure 2.1. The abacus is a physical computer not only in its computation, but also in its input and output.

The earliest machines designed as engines for calculation did not try to deny their physicality. They wouldn't have because they were purely mechanical devices. The abacus, from about 600 BC, for example, encodes numbers entirely spatially. It is programmed as it is read, in position. Here there is absolutely no abstraction of space. Data space is physical space [Ishii, 1997].

Early computers couldn't help but be spatial. They took up space, and they used the nature and qualities of the physical world to perform their work. This continued to be true as the calculating machines abstracted their input and output away from physical configuration to digital displays, as in Blaise Pascal's mechanical adder of 1640.

The critical shift did not occur until electrical logic became cheaper, smaller, faster, and more reliable than physical switching. Motors and gears gave way to tubes and wires. Suddenly physics, which had been a computational medium became an enemy to be conquered. Computers were too big and too heavy, and things needed to get denser. Initially, computers were made into furniture as in 1957's

IBM 705, which doubled as a desk in order to make its outlandish size more palatable.

Transistors, of course, proved to be the vehicle for shrinkage. As they replaced tubes, computers became objects in space as opposed to defining their own spaces. The rest of this history is common knowledge, how the computer shrank and shrank until we began to fold them up and put them in our pockets. But what does this neutron-star-like compression imply?

First, it puts a clear value-system in place: for computation smaller is better. This seems obvious, but it is not the case for many things—houses and snack food, for instance. There is a clear advantage to a computer that is small enough to carry. And physical space has emerged as perhaps world's primary limited resource. But we never seem to stop the furious miniaturizing, and that has to do with computing power. The outsides of electronics have on whole stopped getting smaller. We have already seen cellular phones hit an uncomfortable level of tininess and bounce back somewhat in size. Things that are of the body must remain proportionate to it, but the computational cores of electronic objects are not bound to the body. If they are, it is only as added weight to be minimized. The parts of computation that are necessarily human-scale are the points at which the machine meets the user—input and output. So there is a tension introduced as the limits of human physiology keep computers from spiraling into nothingness, but at the same time we must keep making the insides smaller so that the objects themselves can become more powerful.

No one feels this tension more acutely than the electronics hobbyist. Traditionally integrated circuits, the bread and butter of any reasonably



Figure 2.2. The IBM 705 from 1957.
The computer as furniture.



Figure 2.3. Not so long ago computers made their own spaces. The 1951 Whirlwind computer.



Figure 2.4. The SG2200 from Sewon claims to be the smallest cell phone.



Figure 2.5. Current phones are larger than they were. Now they hide behind large color displays.



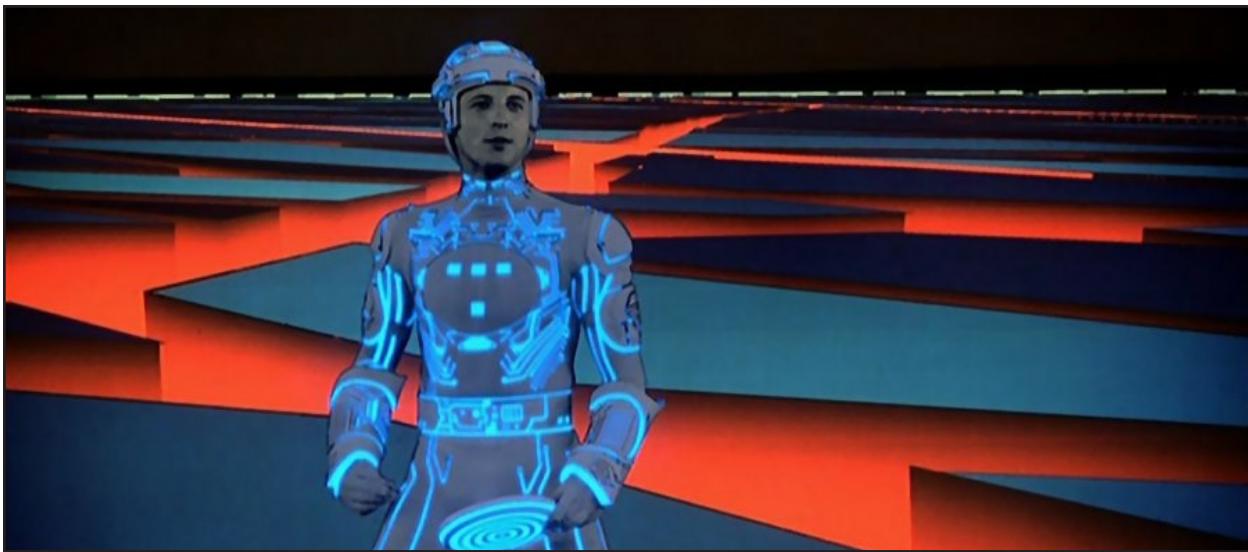
Figure 2.6. Cygnal proudly offers us the C8051xxx microcontroller family. Good luck soldering that one. [<http://www.cygnal.com/>]

complicated electronics project, have been available in packages of sufficient size to allow them to be handled with fingers and soldered by hand—DIP “dual inline packages,” for instance. But many of today’s technologies such as BlueTooth are available for use only in packages with leads so many and so small that no human being could reasonably expect to manipulate them. These types of chips are produced for companies who design circuits on a computer and then have them assembled by robots. This happens, of course, because the economics of serving a hobbyist population doesn’t justify the expenditure. But there is the feeling that consumer electronics technologies are shrinking away from accessibility to individual experimenters.

The physical shrinkage of the machine manifests itself as an embarrassment of the flesh. The thinner the notebook computer, the better. Computation is an anorexic industry. As Katherine Hayles understands it, we owe this desire for the eradication of the body to

a conceptualization that sees information and materiality as distinct entities. This separation allows the construction of a hierarchy in which information is given the dominant position and materiality runs a distant second... embodiment continues to be discussed as if it were a supplement to be purged from the dominant term of information, an accident of evolution we are now in a position to correct. [Hayles, 1999; 12]

Spatial computing proposes to celebrate corporeality of data rather than trying to deny it.



2.1.2 Space in the Machine

Our fascination with the space inside the machine is not new. The Aristotelian universe was essentially a mechanical system that described planetary motions as part of a giant machine. Describing life inside space stations and bubbles large enough to hold populations has been the bedrock of science fiction for as long as we've had it. In 1964 Archigram even reimagined the city as a huge walking robot that could dock with other cities.

At least as far back as the Renaissance, artists such as Durer used machines to help them represent space. In the second half of the twentieth, however, the growing internal power of machines began to allow them to represent spaces and objects directly to our eyes, often spaces and objects with no referent in the real world. Computers turned out to be masters of perspective and simple shading, a few of the artist's simplest tricks for conveying depth. Suddenly there appeared to be whole open landscapes inside the machine.

And as the outsides of the machines shrank and the “space” of memory and storage inside exploded, it became possible to popularize the idea of moving

Figure 2.7. The 1982 film Tron demonstrated the cultural fascination with and fear of being swallowed by the machine.



Figure 2.8. In 1959, the DAC-1 (Design Augmented by Computers), developed by General Motors and IBM, became the first interactive 3D computer graphics system.

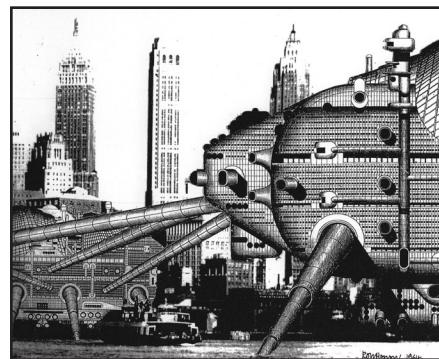


Figure 2.9. Archigram's Walking City, 1964. [Herron, 1964]

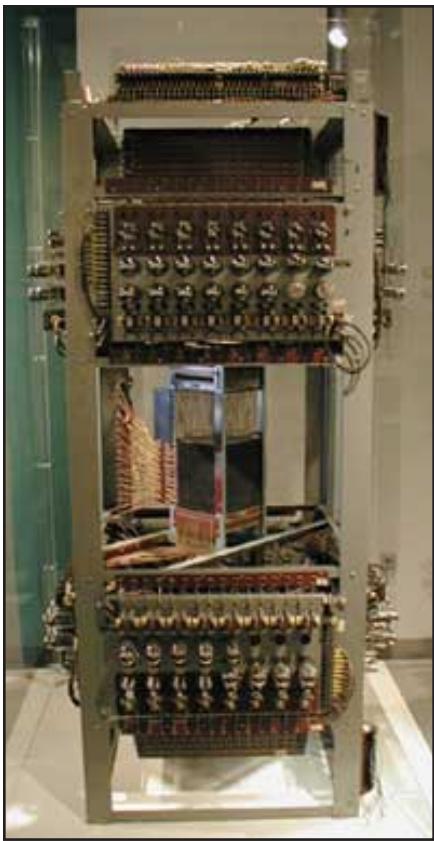


Figure 2.10. 4,000 bytes of memory from the 1951 Whirlwind computer, standing roughly 9 feet tall. Today we put 1,000,000,000 bytes on a single chip.

ourselves wholesale out of messy old real space and into virtual space. A magnetic core memory of 4,000 bits weighed tons in 1951, but now (April 9, 2003), we store a billion bits on a chip the size of a fingernail. The scarcity, expense, and imperfection of real land made the idea of a boundless internal landscape too tempting to resist. This notion was also greeted with anxiety as demonstrated by movies such as *Tron* and *Lawnmower Man*, in which humans are sucked into and trapped inside a virtual environment.

Early computer-generated spaces tended to be (and still often are) rigidly planar expanses of exaggerated linear perspective. Lines are straight, corners are perfect, and ornamentation is minimal. Interestingly this represents something of a return to Modernist form. In 1908 in an essay entitled “Ornament and Crime,” the architect Adolf Loos wrote, “The evolution of culture is synonymous with the removal of ornament from utilitarian objects” [Loos, 1908]. This spare, functionalist notion cast a long shadow over architecture. Mies van der Rohe’s architecture, for instance, exhibits what he called “universal space” and the “open plan.” It results in floating planes and broad



Figure 2.11. A typical representation of a machine-generated “space” from the soon to be released online environment, *Second Life* [Linden Labs, 2003].



Figure #: The Barcelona Pavilion by Mies Van der Rohe. One of the few real spaces that looks virtual.

gridded plazas. Lev Manovich also finds a return to a kind of “soft modernism” in the aesthetics of the web [Manovich, 2002].

The functionalist rationalization of pure forms (“Form follows function” [Sullivan, 1918]) paved the way for Le Corbusier, in many ways the father of Modernist architecture, who famously called the house, “a machine for living in” [Le Corbusier, 1923]. This was possible to espouse at the turn of the century, but could not survive the deconstructive assault of Derrida and others that followed World War II: “It is now a familiar story how deconstruction exposed the inability of systems to posit their own origins, thus ungrounding signification and rendering meaning indeterminate” [Hayles, 1999; 285].

Further, these “functionalist” spaces, pristine in their first realization, did not stand up well to weather and time. They were extremely difficult to build and maintain. Interestingly it is exactly their ease of production and maintenance in machines that keeps them present as virtual architecture although they had faded from prominence in physical architecture before the first computer graphics arrived.

What this really serves to demonstrate is that form follows economics of production. Computers make it cheap and easy to make clean corners, so that’s what we see. Baseboards help cover up irregularities in physical meetings of wall and floor, so most real buildings have them. That virtual environments are becoming more detailed and more topographically complex is due to improved tools for their construction and deployment. There seems to be little ideology driving the development of a virtual “style” except for the quest to do whatever technology has made newly possible. See

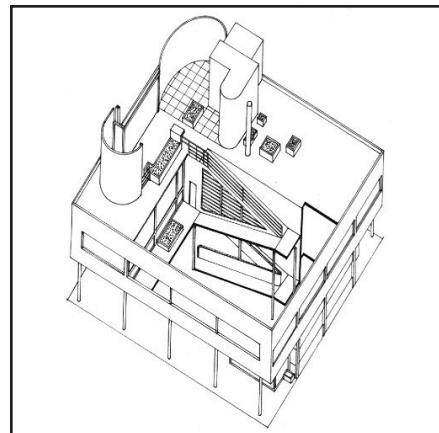


Figure 2.12. Villa Savoye a Poissy by Le Corbusier, who famously called a house “a machine for living in.”

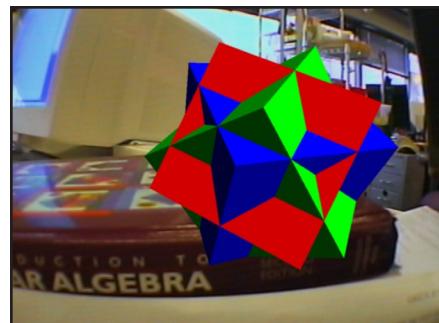


Figure 2.13. It is obvious the floating object is artificial because its colors are too consistent, its lines and corners too sharp.

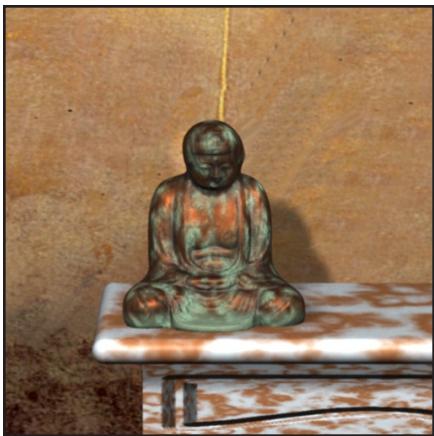


Figure 2.14. A bronze Buddah rendered with a procedurally-generated patina. [Dorsey, 1996]



Figure 2.15. Solutions such as this may have difficulty catching on.



Figure 2.16. Robert Venturi and Denise Scott Brown see the world as a layering of signs and symbols. [Venturi, 2001]

for example the web sites of makers of graphics hardware, such as nVidia [www.nvidia.com]. The features of the cards drive the aesthetics of virtual environments.

One of the hallmarks of the unreality of virtual spaces is their over-perfection. On a computer screen, things look like their ideals (or at least anything with corners and flat faces does). A line is a line and a cube is a cube. These images are unconvincing because we know that there is no real substance looks so perfect. Much time and attention in computer graphics nowadays goes toward making things look imperfect enough to be convincing [Dorsey, 1996], [Paquette, 2001]. It is a hard problem, and it isn't yet solved.

Computer graphics' primitive appeal to an impossible purity makes the idea of virtual space feel somewhat immature and naive, and its throwback to long outgrown architectural ideologies doesn't help either. The proponents of virtual environments have suggested without irony that we use systems that make us look like cyborg monsters. There really isn't anything appealing about this vision to many important sectors of culture. All of this leads to some deserved ridicule surrounding the field of virtual reality.

Where computer graphics diverges completely from spare modern spaces is in matters of graphical interface. The collapse of Modernism brought forward the dominance of the symbol. Architecture blossomed with overt historical quotations and references. Robert Venturi and others recognized that there is no form that does not carry infinite layers of meaning. What is suggested is as real as what is physically present. This is the language of graphical user interface, where the icon reigns supreme, and language is larded over the top of everything.

This mess of signifiers is pretty much where software spaces remain today. Spatial computing does away with icons, lists, and menus as much as possible, to allow things to stand for themselves.

2.1.3 Networked Space

The advent of the Internet considerably complicated the relationship of computation to space. Suddenly connections made inside the machine had the potential actually to span half the globe. Every screen became a portal onto the same shared, parallel world.

The bright side was the promise of an end to solitary virtual existence, replaced by virtual networked communities. And it is true that much of Internet traffic consists of e-mail and instant messages. However, one of the strange qualities of web space is that the user is always alone in it. No matter how many other people are looking at the same information, no one sees each other. Everyone has the feeling of having the entire vast Internet to themselves.

People saw the expanding World Wide Web itself as a kind of virtual space, and it did take root even if it didn't replace the physical world as many feared. It seemed that the Internet could act as a kind of spatial prosthesis, a vastly enhanced telephone. Everything on the web is a single address away—a click, maybe two. (Advertisers consider three clicks to be an unacceptably long “distance.”) But what is the product of total equidistance if not collapse into singularity? It's not a new spatiality, it's a non-spatiality.

The spatial analog of the hyperlink would be the teleportation terminal. Such a terminal opens a

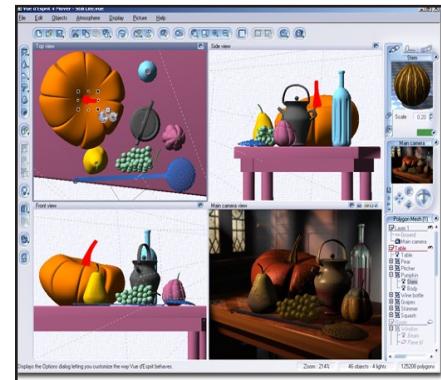


Figure 2.17. Venturi and Scott Brown's vision of architecture makes the world look a lot like software interface.

hole in space, a discontinuity which we cannot contextualize in actual space specifically because it is not of real space. If the essence of spatial connectedness in a network is supposed to be the hyperlink, we are building a space out of holes in space, which is like trying to constitute a sponge out of its voids.

A further problem of networked space results from its devaluation to the point that it can be minted practically for free. Space becomes valueless. As soon as some space becomes ruined, we can just make another new one twice its size. There is no limiting condition to generate value. Perhaps what makes the Internet valuable is that it is non-spatial and attempts to introduce space to it may be fundamentally flawed. (I will have more to say on that in my analysis of my own attempt to do this, *Internaut*.)

2.1.4 The Denial of Space

The Internet is not the only agent of spatial denial in computer science. The dream of escaping the imperfect and unpredictable real world is the engineer's heaven. It is a denial of heaviness, friction, death, and decay. The memory spaces of computer science are the site of huge projects in idealized engineering—where programmers construct machines of astonishing complexity in the absence of gravity and corrosion.

Escape from the uncontrollable and capricious real world into a perfect world of rules, where every consequence has a cause if one knows enough to discover it helps explain the motives of strange hackers and virus writers who measure their success by the quantity of their access and breadth of their spread. These people, powerless in the real world, are masters of the machine, perfect in its

willingness to do as it's told. The author can attest that this kind of power can be very compelling to a young person who longs for an ordered world in which every problem eventually yields to logic. It also helps explain why virtual spaces have had only one resounding area of success—violent first-person games in which players shoot each other at will. These scenarios appeal to the same crowd of teenage boys.

Absurdity grows like a barnacle at sites of cultural tension. All it takes is a look at the size and complexity of the heatsinks that accompany any modern microprocessor to know that engineering is engaged in a fight with physics. We are poised at a point of extreme tension in the spatial relations of computation. I propose a computation that embraces the machine as a spatial object at the same time integrating it with the space inside itself. Spatial computing shares an aspiration with Katherine Hayles's vision for "posthumanity":

my dream is a version of the posthuman that embraces the possibilities of information technologies without being seduced by fantasies of unlimited power and disembodied immortality, that recognizes and celebrates finitude as a condition of human being, and that understands human life is embedded in a material world of great complexity, one on which we depend for our continued survival [Hayles, 1999; 5].



Figure 2.18. The size of this heatsink relative to its host is a sign of the heroic struggle of technology against the physical world.

2.3 The Problems With Realism

Something that tends to go unchallenged is the realism of virtual spaces. The increasing power of processors and graphics cards enables more and more accurate modeling of the physics of light and the mathematics of surfaces. As Lev Manovich understands it, realism has become a commodity we can pay more to buy more of [Manovich, 1996]. But there is a subtlety that is missing from all of the marketing and analysis of virtual systems.

There is a tremendous disconnect between screen-based representations of reality and experiential reality that makes increasingly accurate physical simulations somehow less engaging than it seems they ought to be. The computer graphics term for rendered realism is “photorealism,” and that hints at the problem. The realism that computation tends to aspire toward is the realism of a photograph. Human beings do not experience a photograph as an instantaneous and engaging reality in which they are part. They do not imagine the camera’s eye to be their own. They remain firmly outside the image, and understand it usually as a captured moment of objective representation. It is undeniable that there is something compelling about the asymptotic approach to photorealism. Increasingly accurate renderings continue to inspire wonder even now that the game of chasing reality has grown old.

But the wonder masks an important distinction that virtual reality denies. The wonder is the wonder that the image was not produced by a camera, not the wonder that the viewer was not present as the perceiver of the scene. There hangs above the discipline a notion that we are just a breath away from producing representations that are sufficiently accurate to fool the viewer into total engagement. It can’t happen that way.



Figure 2.19. The Cornell box is the benchmark for photorealism. Rendered images are compared against pictures taken inside a real box. (This one is rendered.) [<http://www.graphics.cornell.edu/online/box/compare.html>]

This confusion of “realism” is apparent from looking at the use of the term “realistic” as it is applied to computer simulations such as games. Sega’s basketball game NBA 2K3 is hailed all over the Internet as the most “realistic” basketball game ever to be produced. What this seems to mean is that the players bodies and faces are taken from real NBA players and the camera shots look like television coverage of basketball. The view is not first-person from a player in the game, and not even from a fan. Instead “realistic” here means creating television with your thumbs. This could hardly be farther from the reality of a player in the game.

This is again evident in the popular, “behind your own back” view in first-person games. It is often possible to switch the first-person viewpoint, which is supposed to correspond to the player’s eyesight to a view that is over the player’s own shoulder or behind them. This is often more convenient for game-play because it shows the player in the context of the scene. The movement of the camera from the virtual eye—the most privileged viewpoint imaginable—to some arbitrary but convenient location outside the virtual body ought to induce extreme disorientation and shatter the vital illusion. But it doesn’t. It is surprisingly easy to digest, and requires no special stretch of the imagination to accommodate. The ease with which this translation is accepted indicates that the egocentric view does not lay special claim to the engagement of players as if it were their own eyesight.

This has everything to do with the nature of perception. The fundamental discovery of art and the physiology of perception since the Renaissance is that the eye is not a camera. Vision is a constructed sense. We have a tiny area of acuity with which we constantly and actively scan the



Figure 2.20. Sega’s *NBA 2K3*. Widely touted as “the most realistic basketball game ever.” [http://www.epinions.com/content_85992509060#]



Figure 2.21. The view from behind yourself in *One Must Fall Battlegrounds*. [<http://thegamebd.tripod.com/previews/OMF/Omf.htm>]

world. Any notion of a photographic experience of a real scene is one constructed by the brain. This is different from the experience of a photograph, which appears as a small colored patch in our field of view. We can understand it as it relates to our experience of the visual world, but it does not mimic our experience of it.

There is nothing “natural” about a rendered perspective projection. It is intelligible, but it isn’t how we see things. In some cases, increasingly “realistic” representations only serve to alienate us from what we are seeing. For instance, in the Quake II engine from Id Software, as the protagonist walks, the view bounces up and down. It is particularly noticeable when the character is walking close and parallel to a textured wall. It is a bizarre sensation to watch the representation of space bob up and down as the player moves forward. But if one looks closely at walls while walking in the real world, it actually does the same thing. We just filter it out so we don’t even notice it. In our minds, walls don’t bounce. So which is the more “realistic” representation? There is a perfectly valid argument that whatever alienates the viewer less is the more realistic. Game players say that after a while one ceases to notice the bouncing, just as presumably, we cease to notice it in the world because it is always present. But I expect that learning to ignore this effect is the same kind of learning that allows players to meld their being with a paddle in Pong. They simply ignore the clear signals that tell them there is an other reality outside of this small area of focus as if it were not the case.

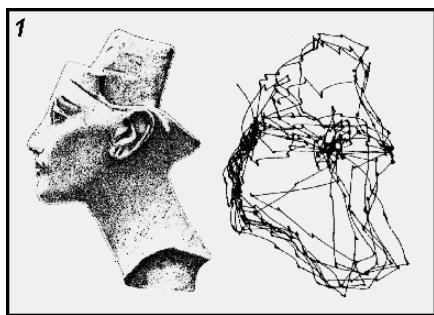


Figure 2.22. Eye-movement traces as a subject explores a picture of the bust of Nefertiti. [Yarbus, 1967]

E. H. Gombrich points out that vision proceeds not as construction of a literal image but as progressive hypothesis testing against actively acquired percepts [Gombrich, 1969]. We refine

our understanding of the world by actively testing it with our eyes, which are attached to our heads. That means if there is an uncertain condition to our right, we may turn our heads. Any visual information we gather there is added to our mental image of the scene in front of us, but the image is as much constructed in reverse from beliefs and memories as it is from light hitting our retinas. In fact patients with lesions in their visual cortex called scotomas, which make it impossible to perceive anything from large patches of their visual fields, often fail to notice they have a problem [Pessoa, 2003]. The process by which the brain stitches the hole with pieces of what it thinks ought to be there is called “filling-in,” and it is an area of active research. In fact we all such a blind spot 10° off the center of our visual focus, but we never notice it unless we design a specific test to discover it. A photographic image may well describe our mental construction of what we perceive, but it vision requires quite a bit of editing to produce what we “see.”

A photograph has done all the work of perception for us. In its pure instrumentality, it supersedes the eye. We are often inclined to trust it more than our vision or our memories. It sucks the subjectivity out of seeing, and allows us to revisit an instant (a finer time slice, by the way, than was ever possible for our eye) at our leisure. We scan the photograph using the same mechanisms of active perception we use in the world, and certainly this helps us focus on different parts of it, but it does not help with any ambiguity that is frozen into the photograph.

There are many conditions that appear visually confusing in photographs that could never be in reality. Sometimes a tree appears to be growing out of a person’s head. We almost never get that impression in reality. The active quality of

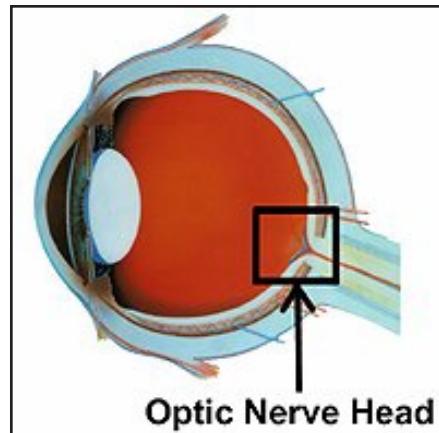


Figure 2.23. Every eye has a blind spot where the retina is joined to the optic nerve. We don't notice it. [www.nidek.com/blind.html]

perception disambiguates the situation before it even becomes questionable in reality. For instance, there is always motion in the real world, and there will be differences in the relative speeds of motion of the tree and the head in the visual field. This effect, called head-motion parallax, is more important to our perception of depth than stereopsis [Arthur, 1993]. Our ability to perceive is distinctly limited in virtual realms because the system cannot possibly respond to all the techniques for active perception that we use. Some systems try to allow for it by using gaze or head-position tracking, but the instrumentality of the system always acts to flatten some of the nuance that we rely on in our sensing dialog with the world.



Figure 2.24. A VR “CAVE” projects 10’ X 10’ images on four sides. [http://www.eds.com/services_offerings/vr/centers_the_way.shtml]

Systems that use specialized hardware to try to replace as much of a subject’s sensory input with synthetic representations of information are called “immersive,” and they all suffer the same unavoidable problem. We have no experience of “immersion” in our real existence. We are part of it, and it is part of us. There aren’t even clear boundaries between the self and environment—it has been contested for centuries. When subjects are “immersed” in a virtual simulation such as a “CAVE,” which projects images on 10’ square walls all around a subject, they have an experience of immersion, which is distinctly not a quality of reality. Immersion is like diving into cold water. One of reality’s hallmarks is that its experience is mundane, and any excitement it contains comes from the thing experienced, not the act of perception.

Paradoxically, the disconnect with reality become most apparent in the most “realistic” simulations. The more the viewer is supposed to be relieved of the effort of constructing a reality out of partial information, the more obvious it is when it fails.

This is why an artifact such as an aliased halo around an otherwise well-rendered character is so disturbing, or why the slightest anti-physical movement is so disruptive. Sidestepping this problem is the considerable work toward “cartoon” renderings of virtual systems, which trade efforts at photorealism for abstraction [Kline, 1999].

Abstraction is a powerful technique that chooses to edit the qualities of a representation consciously away from perfect concert with perceived reality. Such an editing allows for parts of the scene that do not contribute to authorial intent to be stripped away or effaced, and other parts enhanced, offering the viewer only the essence of the communication. Abstraction, like photography, does some of the work of perception for you, but it goes one step further, not aiming to mimic the retinal image, but the ideal image. Of course abstraction can proceed beyond increased legibility into opacity, but this has an artistic agenda more difficult to unpack.

The virtual cartoon dog Dobie T. Coyote from Bruce Blumberg’s Synthetic Characters Group at the MIT Media Lab is a perfect example of abstraction used to enhance communication. His movements and facial expressions are in a way hyperreal since they are a distillation of the visual cues we use to recognize a dog’s mood and behavior. It is important to distinguish this kind of abstraction from metaphor. The dog does not *stand for* a dog simply because he is not “realistically” rendered. There is no system of equivalence established to which the dog as a symbol is appealing. We perceive the cartoon dog as a specific dog and nothing else. What abstraction does necessarily do is universalize, which has the effect of eroding certain particulars in favor of others. However, in the case of a virtual construct that has no referent in the real world, the missing specificity pre-exists nowhere.



Figure 2.25. Screenshot from the AMP II game engine. Almost realistic but not quite. [<http://www.4drulers.com/amp.html>]



Figure 2.26. Dobie T. Coyote from Bruce Blumberg’s Synthetic Characters Group. [http://web.media.mit.edu/~bruce/whatsnew.html#Anchor_new1]



Figure 2.27. Video conferencing facilities are available at the New Greenham Park Hotel. [www.greenham-common-trust.co.uk/ images/video.jpg]

Abstraction successfully avoids the vain attempt to build a virtual simulation with all of the detail and richness of the real world.

This same failure through slavish adherence to maximizing sensory information is apparent in the virtual reality of telepresence, in which a non-present party is brought into “presence” by a virtualizing technology. In all of the telepresence systems I have witnessed, the most obvious quality of the remote parties is their non-presence. The technology that is supposed to bring them closer only serves to emphasize their distance from the goings-on.

Having experimented with webcams for personal connection to help maintain a long distance relationship, I can attest to their inadequacy. (We went back to telephone only.) Often a mentally-constructed or abstracted reality is more compelling than a sloppily constructed representation of a fuller set of sensory information. Readers usually find this the case with film adaptations of books they love.

The inadequacies of “realism” in virtual environments make it worthwhile to look for alternative modes for dealing with the space inside the machine.

2.4 The Problems With Interactivity

Where the problems of realism are problems of space in the machine, the problems with “interactivity” are problems of the machine in space.

There is an irony in the use of the words “active,” “interactive,” and “reactive” to describe computational objects—both physical and virtual. It is a common practice, as though nothing had those qualities until the computer swooped down and started endowing ordinary objects with buttons and microphones. The truth is that non-computational objects are far more active, interactive, and reactive than any working computational version of the same thing. The reason is that in order to consider an object computationally, we must derive data from it, and that means outfitting it with sensors in some way. As soon as we do that, we chop away all of the interactions we have with that object that are not meaningful to the specific sensor we have chosen. No matter how many sensors we add, we are taking a huge variety of interactive modalities and reducing them to several. How could a simulation of a cup ever be as interactive as a cup?

Some argue that adding sensors to a physical object does not constrain its existing interactivity, but augments it electronically. I believe that is true as long as the object remains primarily itself with respect to the user and does not undergo some metaphoric transformation into a virtual representation of itself or into a semantic placeholder. That is difficult to achieve, and cannot be done as long as users must consult a secondary source to determine the quality of their interactions. For users to check a screen or even to listen to a tone to determine the pressure with which they are squeezing an object supersedes their

own senses and reduces any squeezable object to a pressure sensor. In order for a physical object to be augmented rather than flattened by computation, the computation must occur (or appear to occur) inside the object and the consequences of the computation be registered by the object. The object must also not become fragile or restricted in its manipulability.

This challenges the claim of mouse-based Flash authoring to be “interactive design.” It is perhaps as interactive as a phone book, but it certainly isn’t as interactive as an orange. In order for us to design computational objects that achieve that level of interactivity we will have to concern ourselves with more than the screen. The physical body of the computational object is vital to its interactivity.

3. Motivation

My motivation as a researcher stems from a desire to address some of the deficiencies in the current state of our interactions with machines, as detailed above. But there is a personal motivation as well.

My family used to take trips to national parks. These were some of my favorite vacations because I liked to walk inside landscapes that were much larger than I was. I liked to be able to see things distantly and then gradually to approach them and find them to be even more richly detailed than I had imagined them. I was a computer child too, so I often thought about this in terms of resolution and quantization—how the strongest flavors of the real world were due to its infinite resolution. Every pinecone had something interesting to say under each one of its scales if you took the time to examine it with eyes and fingers. No simulation I had ever experienced had that power. They reached only as far as the attention of their creators. But I dreamed of making that simulation. My fantasy was to be able to take flight from where I stood and zoom in any direction to close in at high speed on anything that caught my interest. I would be able to experience it in all of its detail.

The fantasy has faded, but what hasn't left me is a love of the real. What I understand better now are the limits of computation. I no longer dream about producing such a system inside the machine. Instead I have turned my attention to an idea that I think holds more promise: the integration of the real and computed. Rather than try to simulate the qualities of the world I love, why not let the world stand and be present in all its complexity. I have been trying to make systems that engage the physical world rather than deny it.

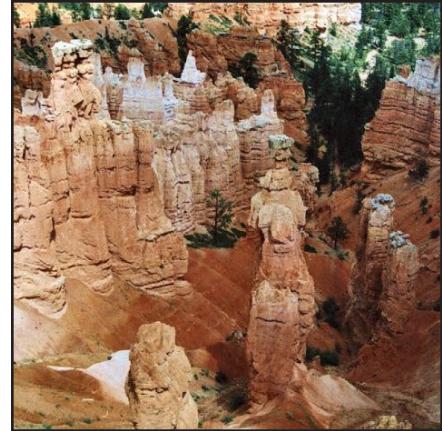


Figure 3.1. I vividly remember Bryce Canyon in Utah. [<http://globetr.bei-online.de>]

Of course, thanks to the house, a great many of our memories are housed, and if the house is a bit elaborate, if it has a cellar and a garret, nooks and corridors, our memories have refuges that are all the more clearly delineated. All our lives we come back to them in our daydreams.

—Gaston Bachelard
[Bachelard, 1964; 8]

“We are continually living a solution of problems that reflection cannot hope to solve.”

— J. H. Van den Berg
[Van den Berg, 1955]

4.

Enter Spatial Computing

Spatial computing proposes hybrid real/virtual computation that erodes the barriers between the physical and the ideal worlds. It is computational design arising from a deep belief in phenomenology. This emphasis delimits it inside the body of HCI practice, the majority of which concerns itself exclusively with what it can measure. Spatial computing is more interested in qualities of experience. Wherever possible the machine in space and space in the machine should be allowed to bleed into each other. Sometimes this means bringing space into the computer, sometime this means injecting computation into objects. Mostly it means designing systems that push through the traditional boundaries of screen and keyboard without getting hung up there and melting into interface or meek simulation.

In order for our machines to become fuller partners in our work and play, they are going to need to join us in our physical world. They are going to have to operate on the same objects we do, and we are going to need to operate on them using our physical intuitions:

Because we have considered the relation between human and machine as instrumental and prosthetic (subordinating and conforming machines to the requirements of human instrumentality), and even more because we have created them in the image of an ideally isolated individual, we have denied our computers the use of the shifters (here, now, you, we...) that might transform their servitude into partnership [Cubitt, 1998; 35].

If we are not already, we are bound to become human beings embedded inside our connected machines. We will be the processors working within the giant spatial networks that surround us. How will we use space, place, and objects to direct that computation?

5. Methodology

My goal at the Aesthetics + Computation Group has been to attack the boundaries between physical and virtual spaces with small incursions from all sides. Therefore my explorations have been many and various in media and scope. Some have been more about place, some more about objects. Each one has led me further toward understanding spatial computing. I imagine each of the projects I developed as a component that could be integrated into future systems that more powerfully complicate the real and virtual than any of them taken singly.

In the course of my study my primary method has been to make things first, and ask questions later. This process privileges intuition over scientific inquiry because it does not produce artifacts designed to test hypotheses. It is an engineering methodology driven not by a functional brief but instead only by the demand that the products have bearing on a broad set of concerns.

I have taken pains as I produced these projects to allow them to change as I made them to take their best course. It becomes clear only in the process of making what a thing's most valuable form will be. This freedom to allow ideas to change as they became real has made my work better. Nothing leads to more tortured and awkward instantiations of ideas than rigidity of purpose.

It was not always clear to me as I worked what the connections between my projects were, and it has required a period of introspection, a reprieve from building, to hear what they have to say. The theory has arisen from the artifacts, not the other way around, and that is the only reason I have faith in it. As William Carlos Williams said, "No ideas but in things."

6. Precedents

Spatial computing is not a new discipline. It is located well within the theoretical boundaries of existing fields and programs of research such as HCI, Ubiquitous Computing, Invisible Computing, and Tangible Interfaces. Each of these is committed in its own way to bringing together human and machine space. All of the experiments I undertook could easily belong to several of these. The distinctions I mean to emphasize between them and spatial computing are small but important differences of focus and in some cases philosophy.

HCI is the umbrella over all of these fields. The interaction between human and machine can be construed in many ways.

Ubiquitous and Invisible Computing

In 1988 at the Xerox Parc research lab, Mark Weiser first articulated the idea of Ubiquitous Computing. It was a call for technology to recede into the background of our lives where we would not even notice its existence:

Our preliminary approach: Activate the world. Provide hundreds of wireless computing devices per person per office, of all scales (from 1" displays to wall sized). This has required new work in operating systems, user interfaces, networks, wireless, displays, and many other areas. We call our work "ubiquitous computing". This is different from PDA's, dynabooks, or information at your fingertips. It is invisible, everywhere computing that does not live on a personal device of any sort, but is in the woodwork everywhere [Weiser, 1988].

Weiser understood Ubiquitous Computing to be the opposite of virtual reality. It admits no possibility for the machine to represent space back to a user. It is the machine dissolved into space. This is closely related to the invisible computer, as espoused by Donald Norman [Norman, 1998]. Norman has trouble both with graphical user

interface (because it doesn't scale gracefully to the level of complexity of modern software) and virtual realities (because they confuse visual information with spatial experience):

These well-meaning but naive technologists confuse what people see when they move about the world with what they experience. If we look only at the visual information to the eye, there is no difference between moving the head to the left while the world stays stationary and moving the world to the right while the head stays stationary. The eye "sees" a moving image in both cases, but our experiences of the two cases are very different [Norman, 1998; 101].

Norman urges a move away from the multifunction PC to the single-purposed "information appliance," whose operation is so obvious that it does not require instructions. He advocates the purging of metaphor from interface.

I primarily agree with Norman and Weiser's critique of existing interface practice. Spatial computing also does not employ graphical user interface (GUI) because it is metaphorical rather than direct. And spatial computing also criticizes pure virtual environments as they are commonly implemented. But it certainly does not advocate invisibility of the machine in every circumstance. And neither does it shun spatial representation by computers. Spatial computing demands only that we be careful with visual representations of space, and rather than harbor vain hopes for their replacement of experiential reality, link them in meaningful ways to existing spaces and objects.

Perhaps the most important difference between spatial computing and Ubiquitous/Invisible Computing is that spatial computing is not polemical. I acknowledge and accept the obvious benefits of GUIs. There is no way that physical reality can offer the kind of chameleon flexibility and complexity that instant metaphor can. I do not therefore advocate the replacement of the PC as it

exists with spatial computing. Spatial computing is inappropriate for tasks such as word-processing, which have no obvious physical connection to begin with. It is well-suited to domains such as design, which use the computer as virtual canvas or clay.

Tangible Interfaces

Some recent and ongoing research at the Lab also shares much with spatial computing. In particular, Hiroshi Ishii's *Tangible Media Group* has an interest in physical manipulation of objects as a medium for computational control. The work of Brygg Ullmer such as his *metaDESK* [Ullmer, 1998], and *mediaBlocks* [Ullmer, 1997] provide a variety of ways to use physical objects and spaces to explore and manipulate digital information. One of the primary differences between what Ullmer and the rest of Ishii's group have done and what I am have been doing is that their work focuses directly on interface. They are willing to use physical objects as icons "phicons." These are objects without previous valence to the user, often abstract blocks or disks. Their manipulation does provide control over a system, but it isn't fundamentally different from software interface except that it is bound by physical laws. They call these systems "TUIs" for Tangible User Interfaces. I think tangibility is important, but it is not my primary concern. Tangibility is a necessary by-product of computational engagement with real objects in real spaces. I would not want to miss it, but I do think that reducing physical object to interface controls unnecessarily saps them of their own identity and autonomy. As Ullmer points out, they are symbolic, standing for something for something other than themselves [Ullmer, 2001].



Figure 6.1. Brygg Ullmer's *metaDESK* uses a variety of physical tools and metaphors to allow users to interact with geographical data.



Figure 6.2. Brygg Ullmer's *mediaBlocks* lets users store and manipulate media clips as if they were stored in wooden blocks.

Unlike Tangible Media, which deals with physical objects as interface, my aim is to obscure and distribute interface so that it becomes impossible to

locate its beginning. Interface itself is unavoidable. It happens at the meeting of any two different media. But in our interactions with physical objects we are seldom aware of interface as such. Our attention extends beyond the interface to the object of our intention. I hope to allow for that push past interface in spatial computing.

A group at the Lab that has done significant work toward embedding computation in existing objects is Joe Paradiso's *Responsive Environments* group. They have placed sensors and computers in objects such as shoes for dance and gait analysis without making them fragile or limiting their use [Paradiso, 2000]. They are also interested in sensor networks, which effectively spread the locus of interface so widely that it may become invisible. Matt Laibowitz is currently defining a "Phenomenological Model for Distributed Systems" (based on the description language *SensorML*), which deals explicitly with issues of active computational perception [Laibowitz, 2003]. These projects go a long way toward integrating the machine into human space.

In recent talks, HCI promoter and researcher Bill Buxton, has expressed concern over the difficulty of transducing objects. We have very few ways to get them into and out of our machines. This is a concern central to spatial computing.

There are also important precedents very close to home. The Visible Language Workshop was the group at the MIT Media Lab that later became the Aesthetics + Computation Group, of which I am a member. They did much of the pioneering graphics work on integrating perceptual depth cues other than linear perspective into computer graphics. In particular some of their research dealt with layering, blur, and transparency [Colby, 1992].

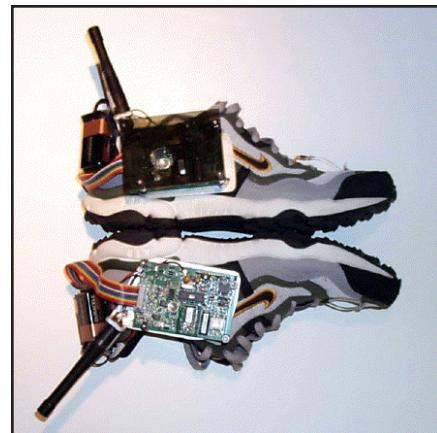


Figure 6.3. Expressive footwear from the Responsive Environments group.

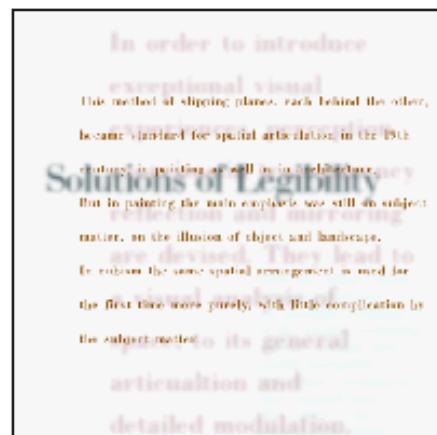
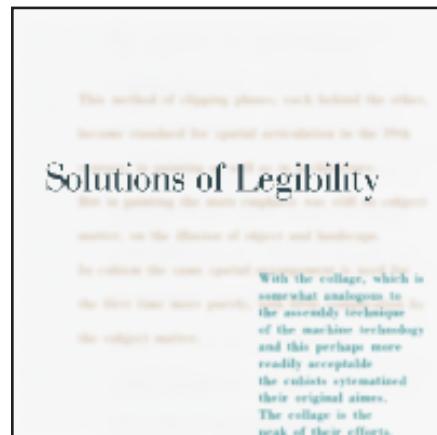


Figure 6.4. The Visible Language Workshop explored layering, translucency, and blur as visual tools.

On the humanist side of this research, Anthony Dunne and Fiona Raby have been looking at ways people react to objects with technological appendages [Dunne, 2001]. For instance they embedded a GPS receiver in a table and had people keep it in their homes for periods of time. They found people became attached to the object and its operation and were concerned when it lost its signal. Some were compelled to take the table outside where it could tell where it was. The attachment people make to active objects is of central importance to spatial computing. The qualities of design that establish that pseudo-empathic relationship are part of what I hoped to engage.

Each of the individual projects I describe in this thesis had its own specific precedents, and those I will detail in their own sections.

7. Roadmap of Explorations

The six projects I describe in this thesis could be organized on several different axes. They could be ordered by their bias toward real or virtual space, or the amount they deal with objects versus the amount they deal with space. Instead I will present them as a chronology because it will give the reader some idea of the circumstances that lead to their conception and the forces that shaped their development.

7.1 Installation

I arrived at the Aesthetics + Computation group after two years studying Architecture at MIT. I was ready to think about computation and space, and eager to explore the resources the group had to offer. Among these was a set of inductive position and orientation sensors called a “Flock of Birds” [<http://www.ascension-tech.com/products/flockofbirds.php>], a surplus flat-panel LCD display that I could be allowed to dismember, and a miniature video camera. I quickly sketched out an idea for a system called *Installation* that would allow users to create and modify virtual sculptures that were visible only through a viewing screen. The viewing screen could be moved freely in space to see the virtual constructs from any angle. This involved the use of two of the inductive sensors (one to use as a 3D stylus, and one to track the position and orientation of a viewing screen); one gutted flat panel; and the camera mounted on the back of the screen. The system took shape quickly and ended up surprisingly close to my original intention. In the end the system allowed users to sketch free-form blobs with the stylus and then install them permanently at any depth into the space of the room as seen through the view screen. When the user moved the view screen, the objects

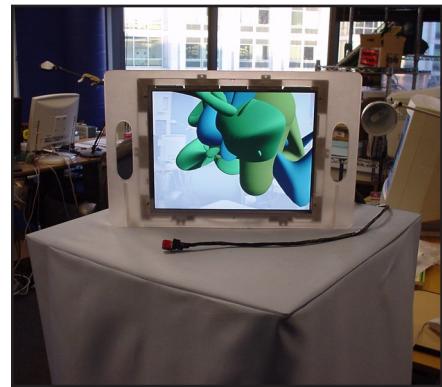


Figure 7.1. The *Installation* setup.

responded as if they were actually in the room. I later wrote an external client for the system, which I ran on several machines around the room. Whenever a user threw an object close enough to one of the clients, it would disappear from the viewing screen and appear on the screen of the client. This gave the strong impression that one had actually flung a virtual object through real space.

7.2 Internaut

After *Installation*, I returned to an old idea that I had wanted to realize for some time—a mapping of the structures of web sites into three-dimensional spaces that could be navigated with a first-person game engine. I guessed that there would be qualities of three-dimensional space that would give some added richness to the experience of navigating the web. After finding a suitable open source game engine, Quake II, from ID Software, I modified it to use maps that I generated from the structure and content of web sites. I called the system *Internaut*. The resulting virtual spaces proved interesting in some regards but nearly impossible to navigate. Users of the system thought of many ways to improve the legibility of the spaces generated, but I think the fundamental flaw was the naïve injection of space into a medium that is fundamentally space-denying. Analysis of this project led me to understand the importance of retaining reference to real space.



Figure 7.2. *Internaut*.

7.3 Stomping Ground

Shortly after this I got the opportunity to work with the Responsive Environments group on a richly spatial installation at the MIT Museum. An old project of theirs, the Magic Carpet, a carpet as musical instrument, was to be permanently installed in the MIT Museum, and they wanted to add a visual component to it. The carpet had a grid

of piezoelectric sensor wires underneath it to sense footsteps and two Doppler radars to sense upper body movement. Users could control the music it made by where and how hard they stepped on the carpet and the overall speed and direction of their body movements. The system had been used in performance by dancers and had had a thorough tour of the world. It was my job to take the same sensor information that Kai-yuh Hsiao had made into music and make it visual. The resulting system, now renamed *Stomping Ground*, used rear-projection to present people on the carpet with greater than life size images of their own legs and feet with blobs rising out of the floor wherever they stepped. In the resulting piece, the space of the carpet was legibly translated into a virtual space in which people mingled with virtual forms.

7.4 Hotpants/LittleVision

After these experiments in screen-based virtuality, my work took a turn toward the hand-held object. I was part of a team that helped teach an undergraduate class in microcontroller design. Our advisor, John Maeda, had us create a development environment from the ground up, which we called *Nylon*. A common problem in elementary hardware design classes is a frustrating bottleneck in actuation. No matter how interesting or exciting student designs are, they are limited in their range of actions: maybe spinning a motor or lighting a few LEDs. We decided to alleviate this problem by building for them a palm-size output device that had significant expressive range. We called the circuit *Hotpants*. It was a grid of 10 by 14 red LEDs each of which could be on, off, or half brightness. We wrote a display language that a microcontroller onboard the display interpreted so that students could send primitive graphics commands to the displays to do things like draw points, lines, rectangles, and circles.



Figure 7.3. *Stomping ground*.



Figure 7.4. The *Nylon* microcontroller teaching platform.

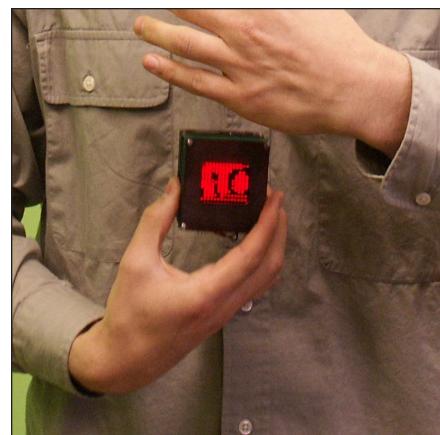


Figure 7.5. A proud workshop participant and his *Hotpants*.

For the purposes of the class, the device served as a display. But because of its size and shape, it was more than a screen. It was a physical entity to be handled and manipulated. Because each pixel was visible, it wasn't possible to forget the physicality of the device and become seduced by the image surface. The image was always teetering on the edge of legibility, requiring the viewer to position themselves in space at just the right distance to make it properly resolve. That said, some projects from the class did not deal thoroughly with the object qualities of the display, and instead tacked it on as though its operation as a display somehow excused it of its body.

But the embodiment of the display was of intense interest to me, and after the class I began to explore it as an independent object. It had its own processor and I supposed it could be used to store and play back small video sequences. I wrote software that allowed image sequences to be compressed and burned directly into the display. This use of the display we called *LittleVision*. Justin Manor wrote video software that allowed us to shoot movies with a webcam and downsample them to the resolution of the device. We ran several workshops in which participants filmed tiny movies using their bodies and large foamcore props. They got surprisingly good results. The most engaging characteristics of *LittleVision* were its size and weight, just large and heavy enough to feel good in the hand. It was a morsel of video, an object to which a person could become attached. Its thingness, its substance in the world was its most important quality.

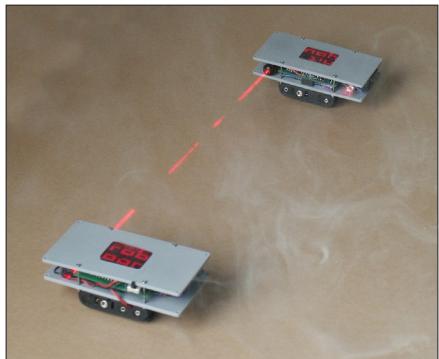


Figure 7.6. The *Word Toss* handhelds demonstrating pointable computing. Smoke provided by Justin Manor. (Note: breaks in the beam here are where the smoke is thin. We don't modulate it at the speed of light.)

7.5 Pointable Computing

As I began to use *LittleVisions*, I started to think about the possibilities and implications of their communicating with each other, which led me to an

analysis of the spatial qualities of different modes of wireless information transfer. It struck me that as the world moves away from wires and infra-red communication in favor of radio-frequency (RF) technologies such as 802.11 and BlueTooth, we are losing the specificity of address that a spatially directed connection offers. It is always possible to tell what a wired device is attached to—just follow the wire. And infra-red devices like remotes are aimable within a fairly narrow cone as is obvious when using a television remote. But RF communications extend almost spherically from their source, making directional intention impossible. We have to resort to selecting the objects of our intentions from a list of names or identifiers. My idea was to emphasize directionality and specificity of spatial communication over all other qualities, and therefore for my carrier of communication, I chose a laser beam, the apotheosis of directionality. I built a system for communication between devices that operates much like an infra-red transceiver, but since it was laser-bound, it was longer-range and totally pointable. This pointability and the feedback the aimer got as a red spot on the object of control were obvious examples of the benefit of maintaining a spatial relationship with computational objects.

7.6 EyeBox

My last experiment, *EyeBox*, went further in the direction of integrating existing physical objects into computation than any of the previous projects. I made a simple 3D scanner out of a collection of inexpensive webcams. I used a technique called “visual hull” reconstruction, which bounds the volume of an object based on the intersection of generalized cones produced from silhouettes of the object taken at multiple angles around it. The technique is described more fully below. It is not capable of reproducing every topography, but it

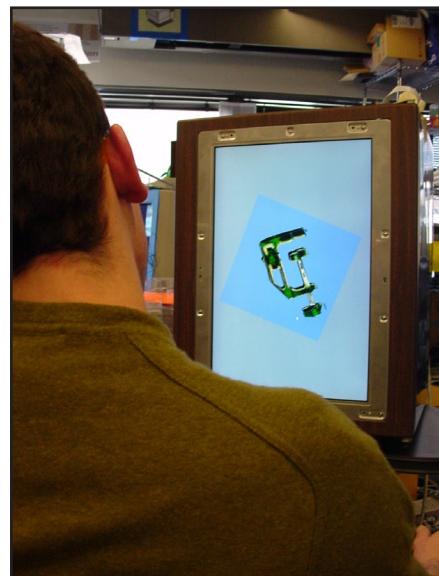
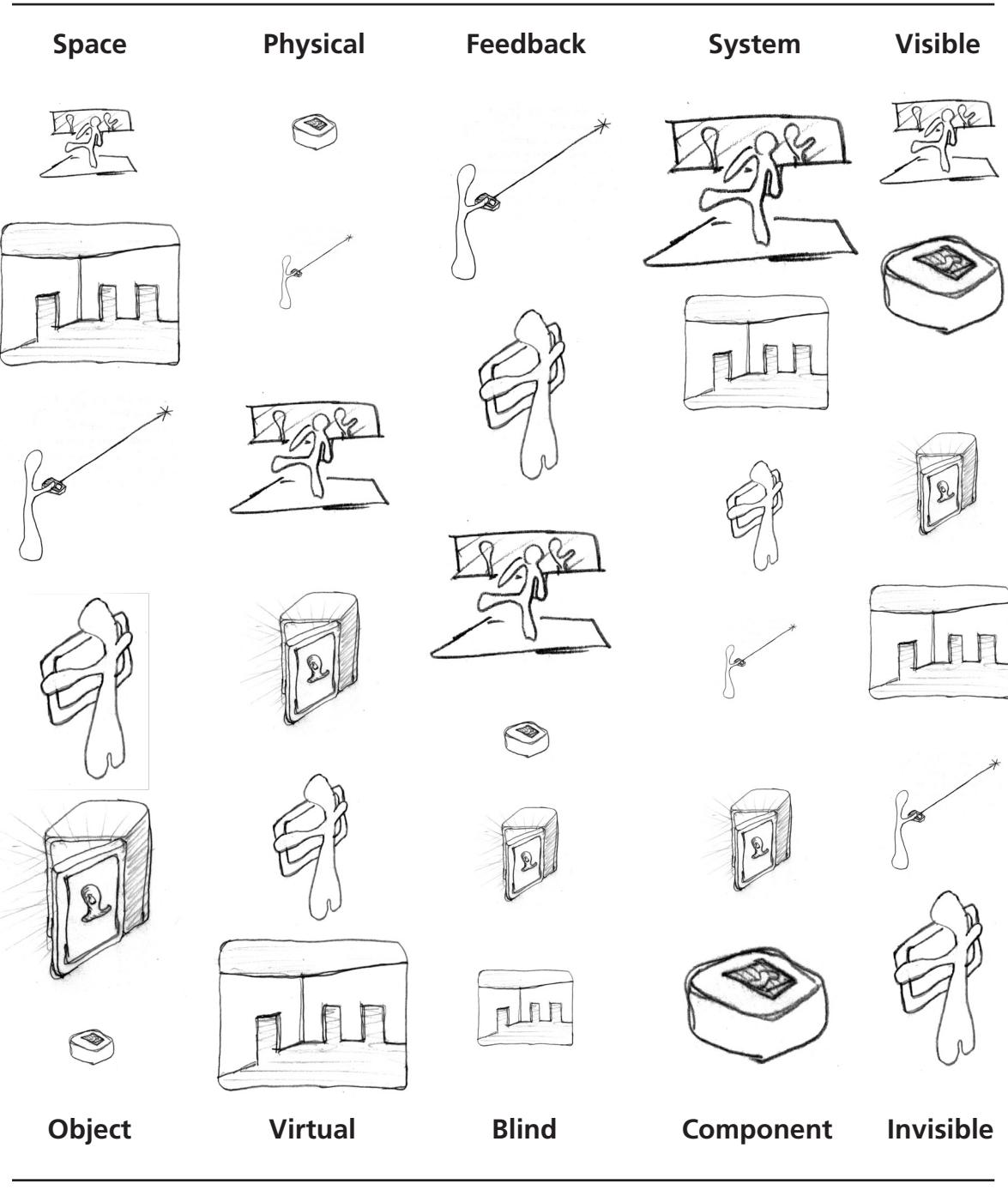


Figure 7.7. *EyeBox* is a mini-fridge turned 3D scanner.

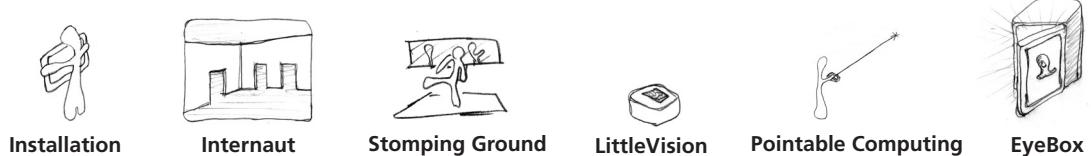
take surprisingly little sensing to produce a very good representation of many everyday objects.

As interesting as *EyeBox* was as a 3D scanner, it was at least as interesting as a model of a new spatial interaction with a computer. The screen in *EyeBox* was mounted on the door of the fridge, and the system was operated by opening up the computer and putting an object inside. The repurposing of the space inside the machine as an active space, not just a cavity containing the guts of the machine engaged people's spatial understanding. It made intuitive sense to them that they should be able to open the machine and put things inside. It was a very pleasurable and complete interaction.

Figure 7.8. The six projects organized along five different axes. They are positioned vertically on the scales of Space/Object, Physical/Virtual, Feedback/Blind, System/Component, and Visible/Invisible. They are sized in the chart according to the amount that the project directly addresses that axis.



Legend:



8. Installation



8.1 Introduction

Figure 8.1. *Installation* allowed users to create virtual forms and install them permanently into real space.

My first project was *Installation*, a system for the creation of virtual forms and their permanent installation into real space. *Installation* consisted of a viewing window and stylus. A tiny camera on the back of the viewing window showed a live feed of the room behind the screen. The stylus and the viewing window were tracked in three dimensional position and orientation to calibrate virtual coordinates with real viewing position. Virtual objects created in the system responded as though they were physically in the space of the room. Once

objects were placed in the environment, they stayed there in perpetuity, pulsing and growing over time.

8.2 System Description

Installation was an exploration in what is traditionally called “augmented reality,” to indicate that rather than trying to replace an experienced reality with a virtual substitute, we are adding to an existing reality with virtual constructs. This certainly qualifies as spatial computing.

Installation presented itself as a cloth-draped chest-height table with a very light flat-screen panel resting on it, which had been liberated from its housing and placed in a translucent plexiglass frame with handles that allowed it to be held and moved in two hands. In the panel, users could see the room behind the screen in a live video feed. This feed was coming from a tiny camera mounted in the middle of the back of the screen. The screen did not quite appear to be transparent, but somehow it was an obvious leap for users to allow it to stand in place of their eyes. Also resting on the table was the end of a long black cord with a half-inch red cube and a single button at its tip—the stylus. When users picked up the stylus they noticed a pencil-like object that appeared onscreen and closely tracked the stylus in the spatial representation. There was no difficulty in understanding this mapping; it was a literal translation of real space to virtual space, and users spent no time disoriented by it or adjusting to it.

When users brought the stylus in front of the screen, a white haze settled over the video feed of the room as if it had become suddenly foggy. The fog cleared up if they moved the stylus behind the screen. The foggy and clear states represented the two operational states of the system, object

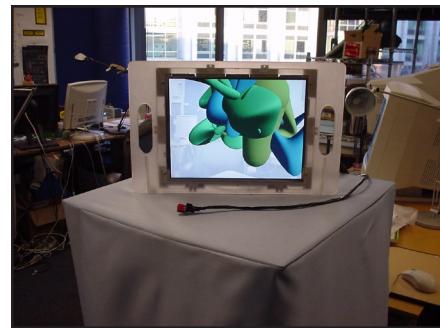


Figure 8.2. The *Installation* setup in context.

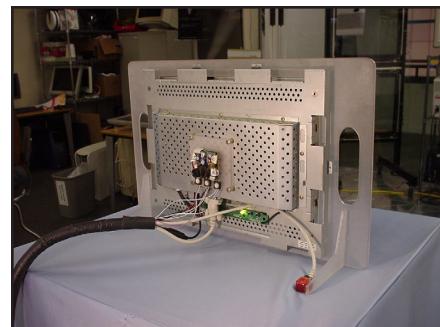


Figure 8.3. The back of the system showing the camera.

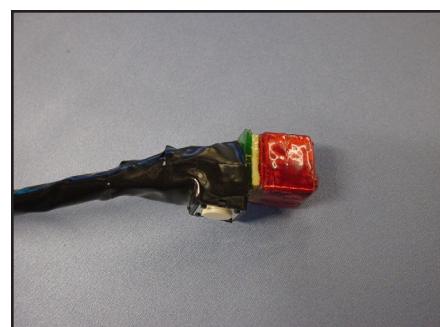


Figure 8.4. The stylus.

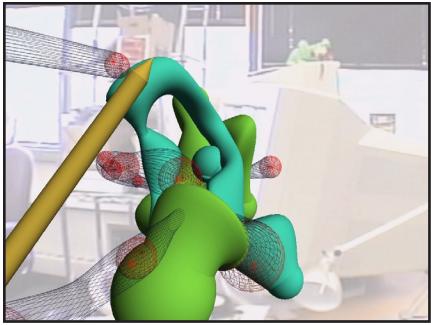


Figure 8.5. Object creation mode. The form tracks the user's gesture.



Figure 8.6. In object placement mode, the user can throw the object into the space of the room.

creation, and object placement. In object creation mode, with the stylus in front of the window, when users pressed the button, a blobby substance appeared to be squirted out from the end of the pencil-like cursor. If users stopped moving, the blobby form continued to inflate. If users moved the stylus quickly, the form was a thin track of the gesture in space, but if they moved slowly, the blob inflated in place, making a thicker form. In this way, users had direct gestural control over virtual forms created in the system. It was easy to make pretzel-like knots or letters this way. *Installation* was not intended as a drafting tool, but a simple gestural sketcher for organic blobby forms. Users could add many separate blobs to a single form by stopping and starting their drawings.

Once a form had been created, if users moved the stylus behind the screen, the pencil-cursor was shown emitting a ray of laser-like red light. This was object placement mode. The orientation of the stylus was tracked, so they could point the beam in any direction they pleased, even back toward themselves. The object they created in creation mode appeared attached to the laser beam a few inches away from the tip of the pointer. Wherever the user pointed the beam, the object followed. When they pressed the button on the stylus, the object shot further down the beam. A grid appeared which helped to show users how far they had cast the object into the scene. Otherwise it would have been very difficult to tell how far away it was since the object was of purely invented form, and its relative size held no information. When users had positioned an object in the space of the room where they wanted it, they could bring the stylus back to the front of the screen, and the blob was left floating in space wherever they had put it. They could then use the stylus to create other forms to further populate the space of the room.

When a user picked up the viewing window, the video feed moved in a predictable way because the camera moved. The virtual forms represented onscreen moved in exactly the way they would if they were truly positioned in space where they were placed. This allowed the user to move the viewing window to look at the objects they had made from any angle, even to cut through them by pushing the window through space they occupied. Through the viewscreen, the objects as seen through the window were fully fledged members of the space of the room. They floated wherever they had been put. In order to add some life to the system I gave the forms the ability to change shape and grow over time. If they were left too long, they grew out of control, filling the space of the room.

The system had no representation of the actual geometry of the room. Therefore the only occlusion that occurred to the objects came from other objects in the system. If a user wanted to place an object a mile away, they could, and at no point would it disappear behind the far wall of the room. This detracted somewhat from the completeness of the illusion. One of the very nice qualities of the system, however, was that it was entirely self-calibrated. That meant that it would work just as well in any space. I did, in fact, show it in a remote location, and it required no special calibration. That movable quality could be important to potential applications of the system, so it would not do to have it interact with a single pre-constructed 3D model of the scene in front of the screen. However, gathering real-time range data and integrating it into the system would be an interesting future effort.

I added a networked client feature to the system, by which objects could be “thrown” to other machines in the room—including the printer. To set up a



Figure 8.7. Moving the viewscreen around causes the forms to react as if they were exactly where they were placed in the room.



Figure 8.8. A client screen (outlined in blue tape) as seen through the viewscreen.

client, I installed the client software, which in its default mode, simply displayed a blank white screen. I then entered a special mode on the master system (the flat panel), in which I placed a sphere into the room directly surrounding each client. I taped blue tape around the border of the monitor of each client, so that a user of the system could identify them. Whenever a user was in placement mode, and they threw an object close enough to a client, it would disappear from the viewing window, and immediately show up on the client's screen, rotating slowly in space. I set up the printer as a client too, and when an object was sent there, it disappeared from the viewing window and got printed out. In this way, users actually had the sense that they were making objects and throwing them around the room.

8.3 Technical details

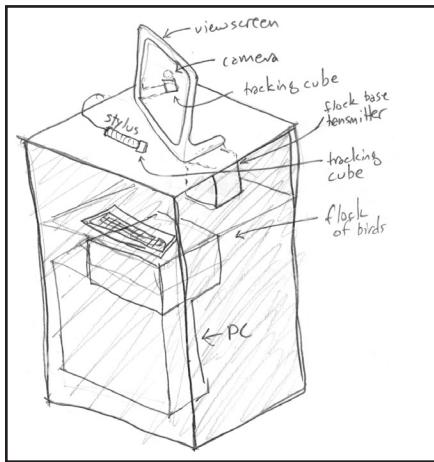


Figure 8.9. System diagram.

Installation was a fairly simple piece of engineering. It had six primary components, the PC, the client machines, the sensing system, the display, the stylus, and the camera. The PC and the clients were totally ordinary Windows machines. The PC talked to the client machines over wired Ethernet. The camera was a small NTSC CMOS camera that went right to a capture board in the PC. The display was a flat-panel LCD monitor with all its housing and shielding removed. Once such an operation is done, a flat panel monitor is a very light, wonderful thing (if perhaps not 100% FCC compliant). It had a laser-cut plexiglass frame surrounding it that had handles for its manipulation. This frame went through two iterations, making it smaller and lighter. The single button on the stylus, and the several control buttons on the back of the display were implemented as stolen key switches from a hacked-

up keyboard—probably the easiest way to get a bunch of momentary switches into a PC.

Sensing System

The sensing system was a “Flock of Birds” from Ascension Technologies, an off-the-shelf inductive position and orientation sensing system. This system itself consisted of three separate types of unit—the signal-processing boxes, which talked to the PC via a serial connection, the base station, and the sensing coils. The base station was placed out of sight under the blue cloth. It was about as large and heavy as a brick. It emitted a magnetic field along three axes that reversed itself at a certain frequency. The two sensing coils, one for the display, and one for the stylus were just coils of wire wrapped in three different axes. By sensing the flux caused by the field in each of the three directions for each of the three axes of magnetic field, the signal processing box is able to reconstruct the location and orientation of the sensor.

Software

All of the software was written in C++ using OpenGL for graphics. Software development fell into three categories. The first software layer processed and integrated data from the sensors, buttons and camera. The second layer acted to calibrate the virtual space to the real space to establish an appropriate projection for the viewing window. The third layer was for creating the forms themselves. I developed a method using spheres connected with Catmull-Rom splines, which provided a fast way to model and render complex organic-looking forms.

8.4 Precedents

ARToolkit

Installation shares features with many augmented reality systems. Some, like AR Toolkit [Billinghurst,

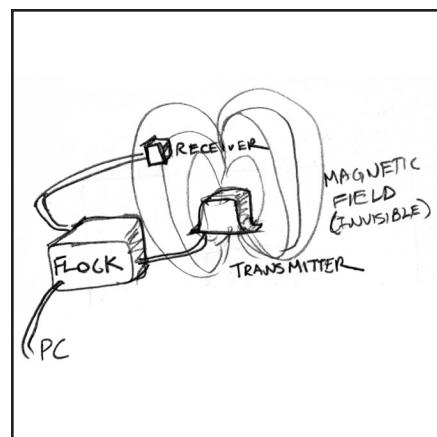


Figure 8.10. Flock of Birds diagram.

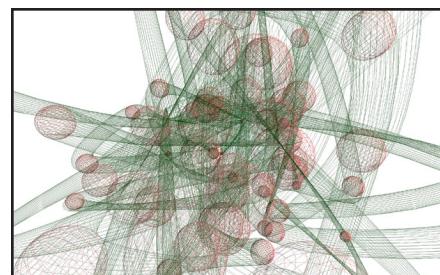


Figure 8.11. The blobby forms were spherical nodes connected with Catmull-Rom splines.

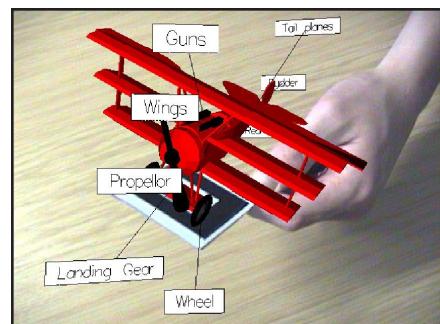


Figure 8.12. The ARToolkit is used to composite a virtual plane into a video image. [<http://www.equator.ecs.soton.ac.uk/projects/arproject/fokker-ar.jpg>]

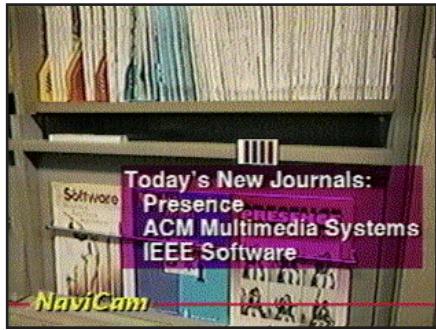


Figure 8.13. Rekimoto's "Magnifying Glass" approach uses a handheld screen to superimpose information. [Rekimoto, 1995] [<http://www.csl.sony.co.jp/person/rekimoto/navi.html>]



Figure 8.14. The Diorama system [Karahalios, 1998]



Figure 8.15. The MARS project at Columbia University implements a mobile augmented reality overlay system. [<http://www1.cs.columbia.edu/graphics/projects/mars/>]

2002], are purely vision-based. They spot known patterns in the world which a user prints out ahead of time. They infer the location and orientation of the pattern by vision algorithms, and then composite previously-defined objects into the scene at the same point. These systems typically act to annotate prepared scenes with prepared overlays. They do not easily allow for creation of new forms or positioning them in arbitrary places in space.

Overlay systems

Many augmented reality systems are used to display information about the world directly onto it as a kind of floating wall text [Karahalios, 1998], [Feiner, 1997], [Rekimoto, 1995]. Like *Installation*, these systems calibrate virtual coordinates to real spaces, but they are quite different in their focus and intent. Augmented reality systems call upon the virtual to annotate the real. Iconic tags or symbols appear overlaid onto scenes to indicate for instance, if there is mail in your mailbox. There is little attention to the forms or space in the virtual, or their interactions with the real, and as a consequence the virtual layer is entirely dominated by the real, appearing as little more than an intelligent heads-up display.

By contrast, *Installation* places more attention on the virtual than the real. If there is a subordinate world in *Installation*, it is the real world, which appears as a reactive underlay for a richer virtual environment. Perhaps *Installation* is less augmented reality than augmented virtuality.

"Eye in hand" systems

George Fitzmaurice seems to have been among the first to describe and develop systems with handheld screens tracked in space. He called these "eye in hand" systems [Fitzmaurice, 1993]. (Interestingly, he used the very same tracking

device I did ten years earlier. It is shocking how little the field of 3D tracking has progressed.) It is surprising, considering that they do in fact map the eye to the hand, how intuitive the “eye-in-hand” model is. This is seen to be a primary advantage of the technique [Tsang, 2002]. Since 1993, there have been several notable systems for augmented reality using handheld screens. One, the *Virtual Car*, by Art + Com, used an overhead armature to track the viewpoint of a screen used to display a highly detailed model of a virtual Mercedes [Art + Com, 1997]. The *Boom Chameleon*, a similarly car-oriented device also uses a hinged rig to track the screen [Tsang, 2002]. This device traces its lineage directly back to Fitzmaurice’s original concept.

There even appears to be a related product on the market, *WindowsVR* from Absolut Technologies in Brazil. Surprisingly, few of the other 3D eye-in-hand systems uses a live camera feed. An exception is a system from NTT, which uses the ARToolkit for vision-based registration [Matsuoka, 2002]. The video feed was one of the most important features of *Installation*, and the easiest to implement. It is possible that they eschewed it out concern that reference to the real world would make small errors in calibration noticeable. My research indicates that people are tolerant, even ignorant, of a great deal of misregistration as long as it is of the right kind.

This list of precedents, most of which I was shamefully unaware of as I produced *Installation*, indicates that this work has a rich history and also an active present.

8.5 Evaluation and Critique

Installation removed the layer of spatial metaphor inherent in most graphical computing by dealing directly in the space of a room. An object created two feet in front of the user was two feet in front of



Figure 8.16. A rendering of Art + Com’s Virtual Car system. [Art + Com, 1997]



Figure 8.17. The Boom Chameleon. [Tsang, 2002]



Figure 8.18. The WindowsVR rig has joysticks to register translation. [Absolut, 2002]

the user. They were free to step around it to operate on it from the side. This kind of readjustment of viewing and working angle is exactly the kind of maneuver that we do continuously without ever thinking about it in the real world, but which we must master some interface to achieve in computational design. As Tsang points out, manipulation of viewpoint in “eye-in-hand” systems requires essentially no new learning. Furthermore, in traditional three-dimensional modeling, operations that change the position of objects viewed through the screen, implicitly change our physical position relative to the scene. But since we know that we have not moved, we must imagine that the entire virtual world displayed in front of us has reoriented without the slightest hint of inertia or other true physical effect. It makes the interaction feel cheap and unreal, and separates us from our work.

This problem with the traditional computational representation of space became obvious on watching people interact with *Installation*. They experienced delight that the objects they created behaved the way their intuition demanded they should. There was an immediacy to the interaction, which people had ceased to expect from machines. It is ironic, perhaps sad, that the operations that seemed magical to users of *Installation* are the most mundane features of our real physical lives. That lifting a viewing window and looking at a scene from a different angle was cause for wonderment, bespeaks the distressing inadequacy of typical human-machine interaction.



Figure 8.19. An immersive CAVE simulation. Is this more convincing?
[<http://resumbrae.com/info/mcn01/session3/>]

In the corner opposite augmented reality, privileging the virtual to the complete exclusion of the real are immersive virtual environments. What *Installation* called into question about these systems is whether it is necessary to jettison all

of the richness and intricacy of the real world to create a convincing virtual experience. The ease with which *Installation* aroused a response from its users indicated that there is a sumptuous experiential quality to be gained by embedding a virtual world within a real one.

Forgiveness and relativity

Some human qualities that proved quite consistent over the course of my projects first became apparent with *Installation*. First, it was reassuring to discover how forgiving of certain discrepancies the human sensory system is. This might be expected given the tremendous extent to which our notions of a consistent reality are constructed from fragmentary sensory evidence and expectation. But it was a surprise to me. The linear algebra I was doing to reconstruct the scene as users moved the viewing window was only so good. It corresponded very roughly with what an actual window would see. Yet the illusion was fairly convincing. That had a lot to do with relativity of sensing. We have almost no absolute references for sensing anything. We gauge things entirely relatively to what else we are experiencing at the moment. This can be demonstrated in countless ways. There are color experiments that show that we perceive color values almost exclusively by value relative to the visual field surrounding a point. This is well-known to any photographer or videographer who has to take white-balance into account. We cannot perceive small global shifts in color temperature unless they happen quickly enough that we can compare them to a fresh memory.

I was fortunate also not to be overlaying virtual objects onto real objects, in which Ronald Azuma states discrepancies of 1/60th of a degree may be noticeable. Instead there was a strong separation between the physical and the real objects, and I



Figure 8.20. Three paintings of Salisbury Cathedral by John Constable. They all use a different color palette to render the scene, but they are all convincing.

did not endeavor to tie them tightly to each other. Azuma in his survey of existing augmented reality applications notes that these discrepancies are severely limiting for certain applications like medical imaging [Azuma, 1997].

Feedback

The believability of spatial connectedness was quite strong. Although the screen did not behave exactly as it would physically, it was impossible to say exactly how it was off, and it hardly seemed to matter since the misalignments were predictable, consistent, and could be counteracted by physical feedback. Azuma refers to a phenomenon called *visual capture*, in which any contradictory sensory information tends to be overridden by the visual. This effect was certainly noticeable in *Installation*. Although the physical movement of the screen may not have exactly matched the screen's representation, the visual took precedence, and the discrepancy went mostly unnoticed.

The importance of feedback can hardly be overstated. As Norbert Weiner wrote, many control problems disappear in the presence of a human operator with sufficient feedback [Weiner, 1948]. For instance, how hard should one push a door to open it? The answer is "hard enough." We don't know how hard we are going to have to push a door, so we adjust our own exertion based on instantaneous feedback we feel about whether the door is yielding. Everything is relative to momentary circumstance and experience. The feedback loops inherent in *Installation* were very tight. The control of the 3D cursor onscreen by means of the stylus was one instance. The cursor was easy to control because it followed the hand directly and it provided onscreen visual feedback immediately. In fact, in object creation mode, there was an inherent spatial translation in effect that



Figure 8.21. A translation takes the gesture from in front to directly behind the screen.

took the gesture being made from in front of the screen to behind it. Almost no user of the system even noticed it. An absolute translation became unnoticeable in the face of tight feedback and good relative correlation.

How little it takes

Another observation that became apparent accidentally during the operation of the system (when the camera stopped working) was how much I was getting from how little. All the camera provided was a live video feed of the room to be plastered behind the virtual objects. It was not calibrated or manipulated in any fashion. But the moment it was removed, the system became entirely flat. Even though users could still use the screen to view the virtual forms from different angles, the primary experience of their existing in the room was utterly gone. It was a shock, and worth remembering how powerful a simple live image can be to create context.

Difficulty of depth

The challenge of conveying depth on a two-dimensional medium is ancient. *Installation* added to that discussion the capability to move the display surface through the scene. But many of the traditional problems of conveying depth remained. J. J. Gibson identified 13 different cues we use to perceive depth [Gibson, 1979]. Not very many of them made it intact into *Installation*. Stereo vision, a favorite of many augmented-reality implementations, was gone. In the absence of any physical referent for the shapes, it was impossible to use their relative size in the scene as a depth cue. Almost the only things remaining to use for depth-cueing were occlusion (of the objects with themselves only), surface shading (but no shadows), and relative speed of movement in the visual field. It was this last that proved the most

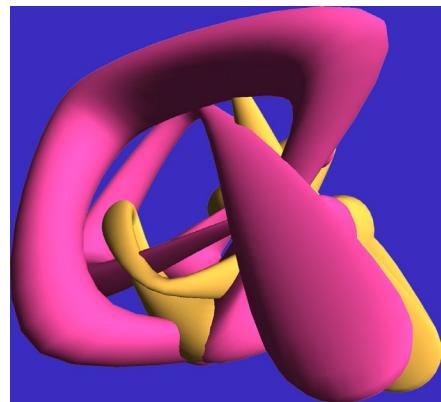


Figure 8.22. Without the background, blobs are just blobs.



Figure 8.23. Georges Braque's Fruit-dish uses many perceptual cues to give a rich illusion of depth without relying on linear perspective.

useful, and the illusion of depth was best when there were multiple objects in the scene at different depths and the user was actively moving the viewing window.

It was interesting also to note how difficult it was for users to draw in an unconstrained 3D environment. They were used to having the structure of a flat surface to press against when making an image. It was difficult for them to control the depth of their drawing. Often if they were drawing letters, for instance, they would be using as feedback only the single 3D view that the stationary viewscreen gave them. So they would close their shapes only to the point of visible closure in a single 2D projection. When they then moved the screen, they would see that their letters went way out of plane and did not topologically close at all. Most letters people drew were not legible from angles different from the viewing angle at which they were drawn. To the extent that this was a failure of the system to translate the spatial intention of the user, I think it should be addressed. What it represents is a failure of feedback. With enough spatial information, users could certainly close their forms. What it would require is a system that allowed for users to change their viewpoint easily as they drew so they could actively perceive their forms. This would probably best be attached to the eye so that head movement could be used in its natural way to disambiguate 3D projection.

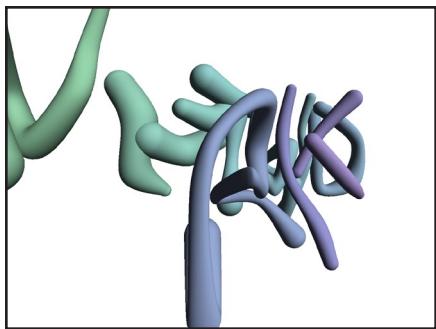


Figure 8.24. These letters were legible from the front. I wonder what they said .

Simplicity

One of *Installation*'s best innovations was a lack of any visible onscreen interface elements except for a cursor. This helped the system to disappear. In particular there were no graphical elements that called attention to the plane of the viewscreen as anything other than a window onto a 3D space. Any buttons, sliders, or text would have set up a virtual

plane that would have been impossible to ignore. It would have distracted from the sense of pure transparency that *Installation* aspired to. Mappings were clear and reactive enough that the systems driving them could be forgotten.

The throwing of objects to client screens was a good example. When a user sent an object to a client, the information to reconstitute the blob on the client screen actually travelled over an Ethernet network. It may have taken a trip to another floor of the building and then back before appearing on the client, but it was invisible and irrelevant to the interaction. As far as experience was concerned, the object flew through the air and landed on that screen.

The importance of this transparency was made obvious by its unfortunate lack in one case. One client, the printer, sat in exactly the wrong place to be aimed at by the system (way in front of the screen, behind the user). Therefore rather than have people throw their objects to the physical printer, I printed out a piece of paper with a picture of a printer on it and taped it to the wall in front of the system as a physical icon for the printer. When people threw their objects to this icon, they printed out on the printer behind them. This separation of the icon from the actual device shattered the illusion of the object's spatial travel, and it exposed the network plumbing underneath it all. Anywhere that metaphor becomes visible, it exposes its separation from the reality for which it stands. It became an important design criterion to avoid metaphor and apparent interface wherever possible.

8.6 Future Work

A Platform for collaboration

The ideas explored in *Installation* become particularly powerful when we imagine several

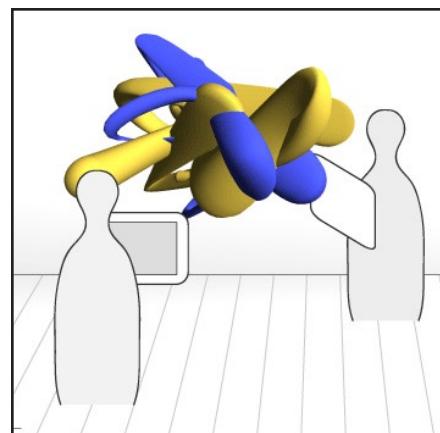


Figure 8.25. Giving forms a shared spatial context allows them to be the objects of collaborative effort.



Figure 8.26. Microsoft Bob suggested the home as a metaphor for information organization. But it took place in a fictional iconic space.

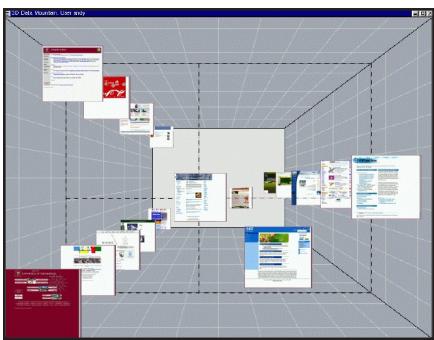


Figure 8.27. [Dourish, 2000] studied storage and retrieval from a spatial model like this. It doesn't have much to say about our experience of real space.

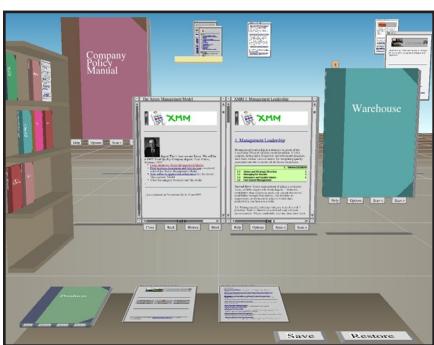


Figure 8.28. Web Forager from Xerox Parc organized web data in a virtual library [Card, 1996].

windows at once looking onto the same evolving environment. Then it becomes a model for luxurious collaborative computation. This model is applicable to any kind of communal form-making, whether it's physical form or abstract information, meaning the ideas could equally find use in architectural design or large-systems engineering. The fundamental idea is that once a work object is placed into space it has a shared context for simultaneous manipulation. This facility is demonstrated by Tsang's system, which he explicitly proposed as a prototype for the collaborative 3D design markup and critique [Tsang, 2002].

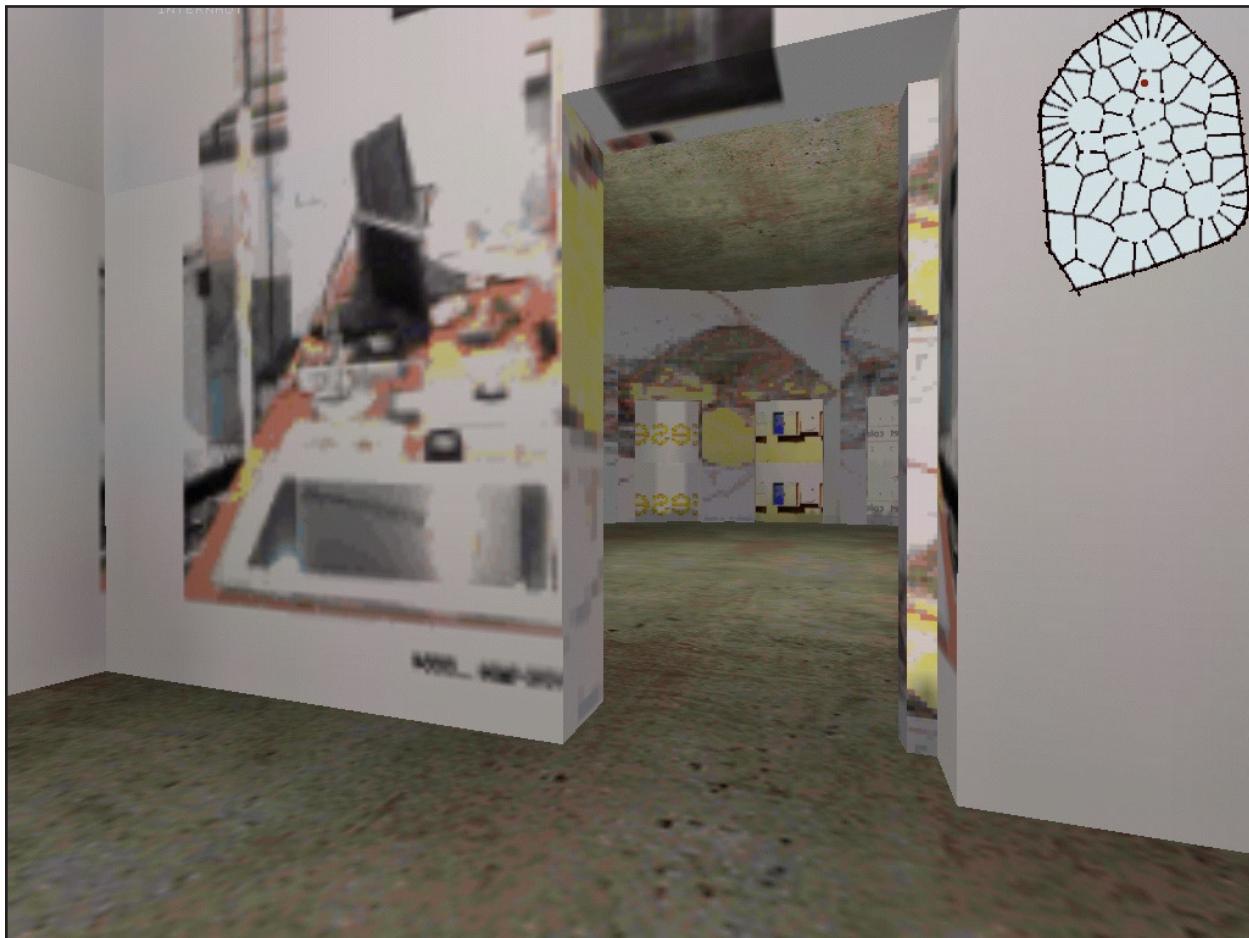
Storage and retrieval

It is easy to imagine the ideas in *Installation* being used for storage and retrieval of information. What could be more natural than to look for something you placed in a physical location? A hierarchy of folders offers very little to the eye to act as retrieval cues. Under most conditions, we cannot even be sure that the position of an item will be constant on our screen. We spend time and energy orienting ourselves to the ad-hoc spaces that the machine tosses at us as fast as we can handle them. Instead why not let the machine orient itself to our own naturally inhabited space?

There have been attempts to apply a physical metaphor to information storage, but few of them have used a real space as the containing envelope. Most of the spaces have tended to be iconic or pure raw regions of linear perspective. I believe neither one has the potential for association that a well correlated real space has.

Installation explores the mixing of real and virtual spaces, and in so doing, begins to fulfill the promise of models for computation that respond to our basic human facilities and intuitions.

9. Internaut



9.1 Introduction

After *Installation*, I turned to a slightly more abstract spatial problem. I wrote *Internaut*, a system for mapping Internet structures into three-dimensional virtual environments and exploring them in a first-person game engine. As such, it did not meet the requirements for spatial computing as outlined above, but was, in fact, a project whose deficiencies were instrumental to my construction of that definition. The analysis of its flaws led directly to my understanding of the importance of spatial computing as opposed to purely virtual environments.

Figure 9.1. A web space made into a virtual space by *Internaut*. A map of the area is shown in the upper right.

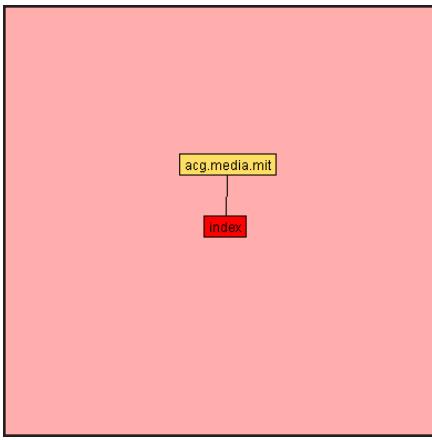


Figure 9.2. A map begins from a web pages and trolls the links on that page.

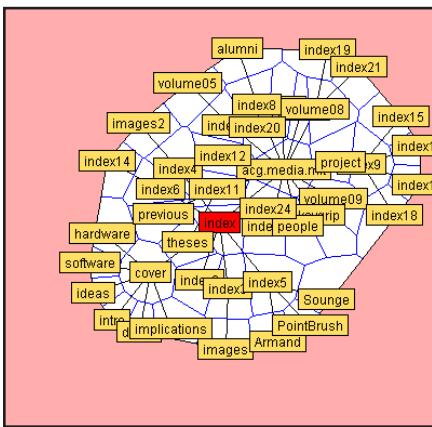


Figure 9.3. A map grows. The root node is shown in red.

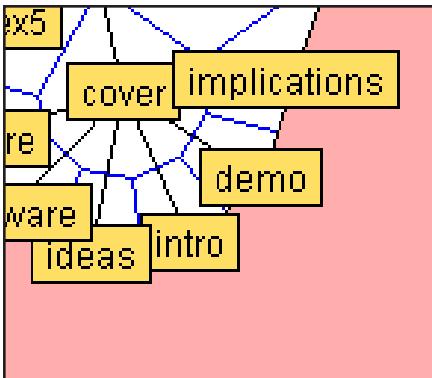


Figure 9.4. In this detail we see that the page “cover” links at least to pages “demo,” “intro,” and “ideas.” These are connected by springs (black lines), which will punch doorways in the walls of the rooms (blue lines).

9.2 Technical Description

The Internet constitutes an enormous electronic architecture that defines places without regard to physical structure. We navigate these spaces with web browsers, moving from place to place with a click on a link. *Internaut* proposed that a physical architecture could be derived from the shape of the network and navigated with a first-person 3D game engine. This was a several-step process, which involved first making spatialized maps from web sites and then processing them into a form in which they could be virtually explored.

The maps were generated starting from a given seed web page by a fairly simple procedure that guaranteed several criteria in the three-dimensional map that I deemed important for them to be meaningful. First, every page from the same site as the seed that was accessible by any path of links should be represented. Second, any two pages that linked together should be immediately accessible to each other. There are numerous ways to design a process to do this, but the one I implemented relied on a simple physics simulation running in Java.

The first page was represented by a node in a 2D graph with a point location. All links on this page to pages at the same site were traversed in order, and these sites were added to the graph as nodes with springs connected to the root node. These simple simulated springs pulled nodes together with a force proportional to their length plus a constant factor for their rest length. It should be no surprise, that these new nodes, which were added to the graph at random locations settled into a ring around the root site. A user was allowed to click and pull on any node in the graph at any time. All springs stretched to accommodate such manipulation, and snapped back into a relaxed

configuration when released. Each new page was then processed in the same way as the root node in the order in which it was added. The resulting network of nodes connected with springs was a stretchy gyrating mass that was constantly attempting to relax into the lowest energy state consistent with its topology of connections.

The nodes were then separated from each other with walls that were the divisions of a Voronoi diagram. A Voronoi diagram associates each node with the area surrounding it that is closer to it than to any other node. This is always a lattice of convex polygons surrounding each node, guaranteeing that each node gets some share of physical space. The springs connecting the nodes intersected these Voronoi-generated walls at many points. Anywhere they intersected, a doorway was drilled in the wall, insuring that any link became a navigable path from one cell to another. This structure successfully located pages in a 2D map close to pages to which they were linked. Obviously there are linking conditions possible in web sites that are not possible to represent in a 2D map with strict adjacency, but the method guarantees that these will be pulled together more strongly the further they are separated, so it does a good job of creating spatial representations of web structures.

The next task was to go from a map in this Java application to a map usable in a 3D game engine. I chose a modified form of the Quake II engine from ID Software because it is now a mature open source project. I generated a map file for this engine with the largest image on any page tiled onto its walls as repeating wallpaper. This surface image was the only distinguishing feature of any room. I undertook extensive changes to the engine to demilitarize it, removing the guns and gangs of monsters bent on killing the explorer, and

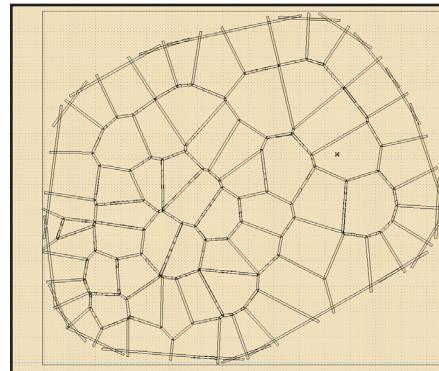


Figure 9.5. The map is then processed in a Quake map editor.

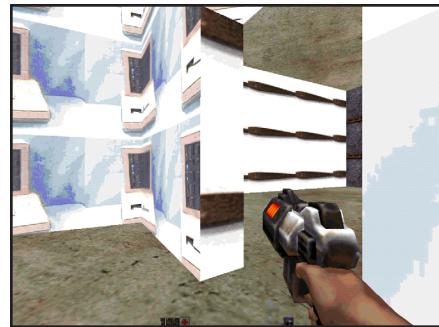


Figure 9.6. I then had to demilitarize the game.



Figure 9.7. After removing the gun and adding a mapping feature.

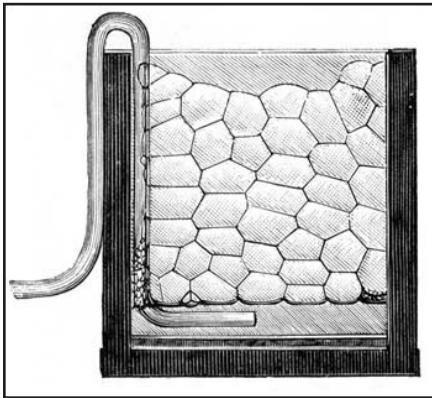


Figure 9.8. Soap bubbles make voronoi patterns [Boys, 1959]. Referenced from [www.snibbe.com/scott/bf/bubbles.htm].



Figure 9.9. Scott Snibbe's Boundary Functions [<http://www.snibbe.com/scott/bf/>]

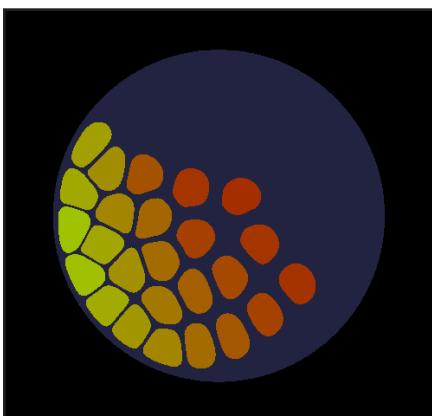


Figure 9.10. Jared Schiffman's honey. [Schiffman, 2000]

adding a mapping feature which displayed a map of the entire site onscreen at all times. I retained the engine's capability to run in a networked mode in which multiple players could explore environment together, seeing each other, and even communicating via typed messages.

I showed the project repeatedly, letting users select the starting web site and then allowing them to navigate the resulting three-dimensional map. As I watched them try to orient themselves to this remapping of Internet space, I became aware of several things that would inform my subsequent work.

9.3 Precedents

For the self-organizing map component of the project, I had many good precedents. This sort of problem has interested scientific and artistic communities for a long time. Voronoi diagrams have broad application to many problems in analytic geometry and self-organizing systems. For instance they can be used to position nodes in self-organizing neural networks [Suanders, 2001]. And they arise naturally in many situations in which surface energies are being minimized as in soap bubbles. They appeal to computational artists and designers for their organic appearance and ease of production. Jared Schiffman used them for their organic visual quality in *Honey*, an exercise in cellular form [Schiffman, 2000]. Scott Snibbe used them for their partitioning ability in *Boundary Functions*, in which participants stepping on a platform are automatically separated from each other by boundaries projected from above [Snibbe, 1998].

Simulated springs are even more commonly used in computational design. They lend movements a

squishy, organic feel. Benjamin Fry used springs to organize web spaces in *Anemone*, which tracks web traffic as a continually evolving network of nodes representing web pages [Fry, 2000]. Judith Donath has used springs to create a self-organizing visualization of human social networks [Donath, 1995].

Ideas of organic form and self-organization have become popular in theoretical architecture in recent years. Greg Lynn uses such forms as “blobjects” in his designs.

Mappings of non-spatial networks into virtual spaces are not new either. In 1996 Apple briefly promoted a 3D flythrough technology called Hotsauce for web page meta-information [Apple, 1996]. AT&T Research produced a system called CoSpace, which used an additional layer of VRML on top of existing web pages to represent web spaces [Selfridge, 1999].

Other networked virtual environments were designed spatially from the beginning. Certainly networked first-person shooter games like Quake III Arena have been successful. It is easy to convene teenage boys in a virtual space with the lure of their being able to shoot each other with impunity. We are currently experiencing a small explosion of nonviolent networked virtual environments that are not meant to represent existing web spaces, but to establish parallel virtual Internet spaces that are constructed and inhabited by a broad public. Examples include the *Sims Online* [Electronic Arts, 2003], *Second Life* [Linden Labs, 2003], and *There* [*There*, 2003]. Several systems like these already exist, but do not find wide use.

A strange feature of the places that users construct in these virtual environments is that they mimic

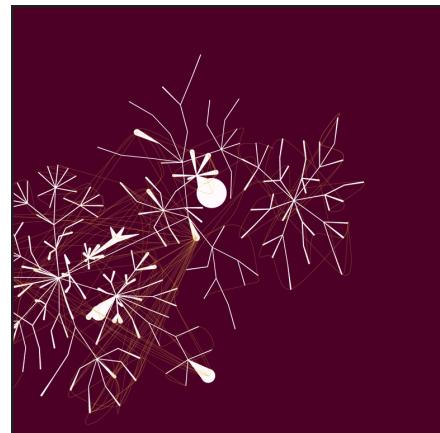


Figure 9.11. Ben Fry’s *Anemone* [Fry, 2000].



Figure 9.12. A study model of Greg Lynn’s.

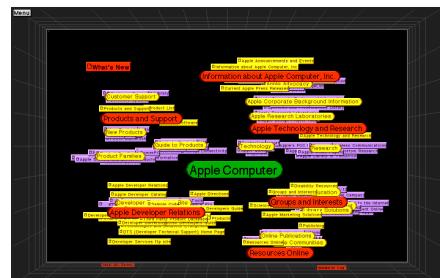


Figure 9.13. Apple’s Hotsauce meta-content 3D web flythrough plug-in. [Apple, 1996]



Figure 9.14. CoSpace, a 3D web browsing system from AT&T research. [Selfridge, 1999]



Figure 9.15. The Sims Online. [Electronic Arts, 2003]

structures in the real world. They put “roofs” on their “houses,” for instance. Why? There isn’t any rain, or in fact weather of any kind to defend against. And enclosure represents no protection from outsiders. It must be a desire for familiarity that drives people to make designs that take no advantage of the liberation that they might experience in these worlds without physical limitation.

9.4 Evaluation and Critique

Users found wandering around the spaces generated by *Internaut* engaging, but also found them confusing and difficult to navigate. Even with the help of a map, they had difficulty finding their way around or remembering where they had been. I think there were several probable contributing factors.

First there was the elimination of all text. I expect people would have difficulty navigating any web sites that had all the text removed and left only a single image to denote each page. Links would no longer describe their destinations in words, but be tiny thumbnail images of the image on the page they linked to. Navigating spaces like this would, I expect be somewhat bewildering too.

But even in the absence of text, there was a difficulty in navigating the structure due to its unfamiliar and inhospitable physical structure. There is a reason that we do not construct our building plans as Voronoi diagrams. The spaces that these generate tend toward spatially undifferentiated rotundas of doorways that make it impossible to identify a dominant spatial axis. Even when there is one, it is not shared by any adjacent cells. Under such conditions, it is often impossible

even to identify the portal through which one entered a space.

We are careful in architectural plans to define circulation space. We do not expect rooms to function both as destinations and corridors for movement at once. The Voronoi plans make no such circulation. There are no clear means of passage between spaces that do not directly abut. To get from one end of the space to the other it is necessary to turn at every room, potentially even away from the final destination. There is no organizing logic that makes the space serve an intention other than aimless wandering.

Use of an organizing geometry other than Voronoi could potentially help this. There are experiments in grammatical architectures that could help point the way to saner structures [Brown, 1997]. That is one possibility for future research. These geometries might allow for the use of more information from the web sites than simple topology. It should be possible, for instance, to identify the primary entrances to the web site. These should represent entrances to the virtual space as well. (In the existing geometry they are most likely to be buried at the center and surrounded by a ring of ancillary pages.) It is likely that some links from a page are more dominant than others—larger text or higher on the page. These should be represented by larger openings or grander access.

Another possibility is that part of what makes the Internet successful is that it is fundamentally non-spatial. Certain conditions of spatiality do not apply to it. For instance there is no such thing as a one-way connection in space. There are doors that lock from one side, but adjacency is commutative. Not so in a non-spatial network. One page may



Figure 9.16. Internaut tended to offer the user a bewildering array of self-similar doorways.

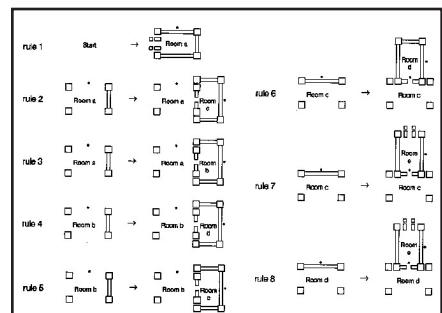


Figure 9.17. Rule-based design from Gero [4.290 Production Systems, Fall 2002].

link to another that has no idea of the existence of the referrer. This network of one-way streets has the tendency to channel users toward sites that are commonly linked to [Barabási, 2002; 57]. These have a higher chance of being useful than the sites that are seldom referenced. There is also a trail of bread crumbs that web-surfing leaves that a user can always use to backtrack via the “Back” button. No such facility exists in real space, although it could be simulated by having movement leave a trace in a virtual environment.

The most damning concern may be that the fundamental property of Internet space is the collapse of distance. Distances are measured in the number of clicks the path takes, and a long one may be three. This implosion of space is necessary to what makes the Internet a useful complement to the real world. An advantage of shopping online is that every store is equidistant at a distance of one click, or the typing of its address. In order to spatialize this condition, it would require a bewildering portal—a spherical mall with thousands of openings that would be a thrilling sight, but hardly useful. It must not be necessary to cross any space to have access to another. Once the intended destination is identified, the need to “walk” there only represents wasted time. Access must be as fast as the delivery of information will allow. So perhaps the idea of a spatial Internet is fundamentally flawed. Cyberspace as Jean Baudrillard puts it is

Where all trips have already taken place; where the vaguest desire for dispersion, evasion and movement are concentrated in a fixed point, in an immobility that has ceased to be one of non-movement and has become that of a potential ubiquity, of an absolute mobility, which voids its own space by crossing it ceaselessly and without effort [Baudrillard, 1988, p. 32].

In a study of the necessity of legibility for virtual spaces, Ruth Dalton concluded that global

intelligibility is not important in systems such as the web where that structure is not used for navigation. Web space developed without any need for an intelligible global structure, and to try to impose one is likely a fool's errand.

9.5 Future Work

Lots of the issues raised in the first part of my critique could be addressed with sufficient further work. We could try to generate rule-based architectures that are more legible and easier to navigate. While I think the program of virtual representation of Internet structures has something to teach us, particularly in using visualization to uncover topologies and flows of the network, I do not think its spatialization for browsing purposes is generally useful.

People suggest that it would be a good shopping interface, in which a user could walk around and see merchandise disposed around a space while talking to others virtually browsing with them. That is a possibility, and I think it would initially be exciting to some, but I don't think its long-term effectiveness would be any greater than nicely displaying merchandise on a web page. The *Sims Online* may well succeed, but I believe that that will have more to do with its nature as a game than as a networked space. Remember that the non-online version of the *Sims* was wildly popular too. I have come to believe that there is more interesting territory to explore in the realm of spatial computing, in which the spaces involved are real spaces that the user already has attachment to and experience with.

10. Stomping Ground

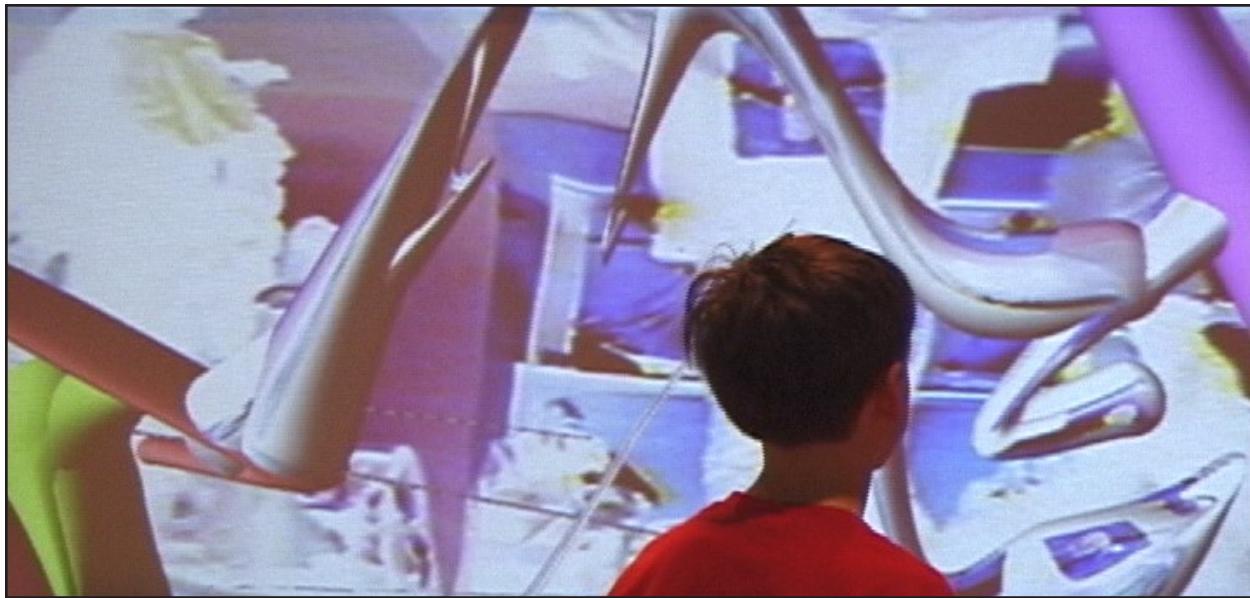


Figure 10.1. A kid engrossed in Stomping Ground.

10.1 Introduction

Stomping Ground is a permanent installation at the MIT Museum consisting of a musical carpet and a projection of live video with superimposed blobs. It is a collaboration between Joe Paradiso director of the Responsive Environments group at the Media Lab, who made the carpet and the radars, Kai-yuh Hsiao of the Cognitive Machines group, who wrote the music, and myself, who designed the space and programmed the visual interaction.



Figure 10.2. Rewiring the carpet with piezoelectric wires. [Photo by Stephanie Hunt].

10.2 System Description

The carpet tracks the location and intensity of footfalls with a grid of sensors. Doppler radars mounted on the sides of the projection wall track the overall direction and intensity of upper-body motion. This information is used to create a musical composition that has two modes: one has a richly layered ambient sound, and the other is

aggressively percussive. The same data is fed to the graphics system, which produces blobs that grow upwards from the locations of footsteps. The blobs are superimposed on a live video image showing the legs and feet of people on the carpet (whole bodies of very small people). The video and the forms in it are warped by a virtual fluid simulation, which is stirred by stomping and upper-body activity.

10.3 Precedents

Prior to my involvement, the carpet had been exhibited as part of exhibits on musical instruments and hosted dance performances. As should be the case in the extension any good work, the past work served as my foremost precedent. I studied footage of these events, the sound and code of the music-making, and the technology behind the operation of the carpet [Paradiso, 1997].

One of the music's modes has a watery background sounds, which led me to give the graphics an undersea feel. I used an intuitive 2D fluid-flow model by Jeffrey Ventrella to warp the projection based on flow induced by "forces" from the radars [Ventrella, 1997].

The blobby forms I adapted from *Installation*, connecting their nodes with springs, and subjecting them to reverse gravity, which pulls them up from the base of the display and out of the picture.

10.4 Evaluation and Critique

It was an interesting challenge to come into a project that already had such a thorough life independent of visualization. I wanted both to fit into the framework as it existed—the expressive qualities of the music, the two modes—but I wanted also to make my portion of the project my own. I



Figure 10.3. Kids exploring the carpet.

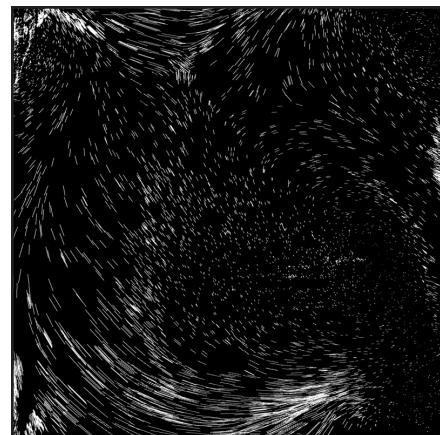


Figure 10.4. I implemented a fluid flow model from [Ventrella, 1997] to warp the video image.

wanted the visual component in the end not to be separable from the whole experience.

Invisibility

Stomping Ground represents an intermediate step in the integration of physical and virtual environments. The real space of the carpet is represented on the screen while virtual artifacts swirl around on top. It is an augmented and distorted mirroring. Unlike the direct and obvious form-making control users have with *Installation*, in *Stomping Ground*, the link between behavior and form produced is less obvious. More was being decided by the system, making the system itself more present as an agent. As much as it was a goal of *Installation*'s to make the system invisible, it was a goal of the *Stomping Ground*'s to become a focus of attention. It was the exhibit as much as the forms and sounds made by it were. In that way it blurred the line between instrument and artwork.

11. Hotpants/LittleVision



Figure 11.1. A bunch of *LittleVisions* running tiny movies.

11.1 Introduction

Hotpants was a handheld display device originally designed for use with the NYLON microcontroller system [nylon.media.mit.edu], which we produced to teach basic microcontroller design to undergraduates. Then as I became interested in the display's potential for autonomous operation, I untethered it from NYLON, renamed it *LittleVision*, and began to use it as a standalone device for the recording and showing of short video segments.

11.2 System Description

Hotpants/LittleVision consists of a very simple circuit which uses a PIC microcontroller to drive four shift registers and two current source

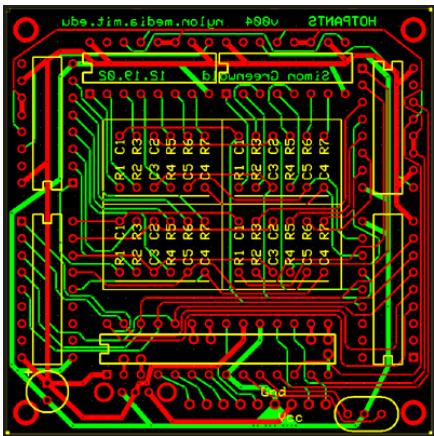


Figure 11.2. The *Hotpants* circuit layout. For a usable version, see Appendix C.

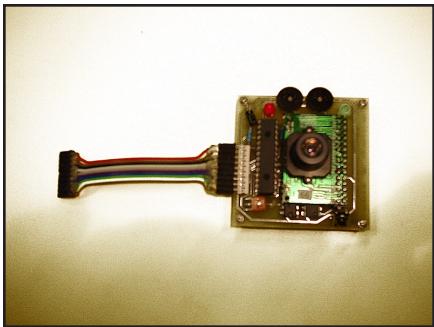


Figure 11.3. A standalone camera board turns *LittleVision* into a self-contained tiny video camera.

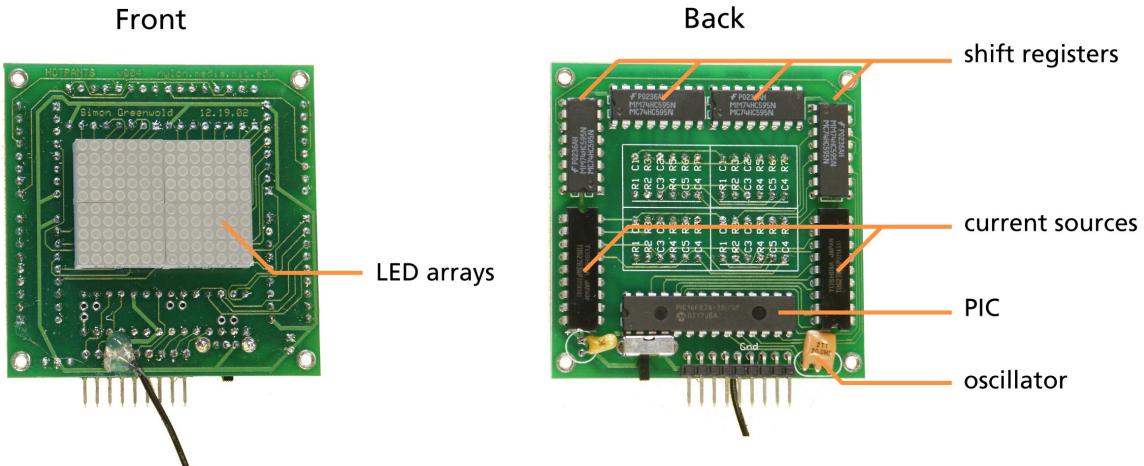
chips, which in turn drive a matrix of 10 X 14 red LEDs. These LEDs can be set to display at full brightness, half, or off. The board exposes a set of programming pins, which are used to connect the board to a PC for downloading of new movies. The board stores about 300 frames, depending on how well they compress, and plays them back at 12 per second, for a total of 25 seconds of video. After this period (or shorter if the movie contains fewer frames), the movie loops. I have recently developed a second board, a camera board, which can be used to record movies directly to the *LittleVision* without the use of a PC.

The circuit and its components are quite inexpensive, and were designed with that criterion in mind. There are much nicer display elements available than these red LED arrays, but they are all more costly. We have run several workshops in which participants film movies of themselves or other props and then burn them to the devices and take them home. In one two-day workshop, we had participants build their boards the first day and make movies the second day.

11.3 Technical Details

Hardware

The whole circuit was controlled by a PIC 16F876 microcontroller running at 20 MHz. It had 22 usable I/O pins. We were using it to drive four 5 X 7 LED arrays. The LED elements in the arrays were referenced by row and column, so we did not have simultaneous unique access to each one. Basically what we had to do was turn on one column at a time and light each row that needed lighting in that column. Then we quickly switched to the next column, and so on. That meant that each column was only lit for a fraction of its possible time. This was sad, as it cut down on brightness, but unavoidable. We did, however, play one nice



trick, which was to treat the four arrays as two tall columns rather than one large array. That way we could control each LED while keeping the columns lit $1/5$ of the time rather than $1/10$, effectively doubling the brightness. (This may make more sense on inspection of the PIC code that drives it. [Appendix C])

Unfortunately, that meant that we had to control two columns of 14 LEDs independently. With 10 columns and 28 effective rows, we were saddled with a burden of 38 outputs, which the PIC couldn't provide by itself, so we used shift registers. Shift registers turn serial outputs parallel by piping clocked values to their output pins on a specific signal. We hooked up 4 shift registers in series, and ended up with 32 extra outputs controlled by 3 pins on the PIC (data, clock, and output enable).

Finally we had a potential problem with constant brightness. We wanted all of the LEDs to be equally bright, but the PIC had a limited ability to sink or source current, which meant that when it was lighting 14 LEDs at once, they would be too dim, and when it was lighting one, it would be too bright. So we ran the PIC column outputs through a Darlington current source chip to give it muscle. On four AA batteries, the circuit ran with intermittent use for about six months.

Figure 11.4. Annotated images of the circuit innards.

Software

There were several different incarnations of software for *Hotpants* because it was used in a number of different contexts. All of the software for *Hotpants* had two components, one on a PC and one on the board. A system by Megan Galbraith allowed a user to write programs in the Nylon language and send them to *Hotpants*. I wrote the setup that lets a user take movies with a webcam and send them to the board. The software on the PC side and the firmware on the PIC is different for each application. We burned a bootloader onto the PIC ahead of time to make it easier to download different programs to the board to change its functionality.

The basic operation of the firmware on the PIC was to change the values in the display buffer over time. That became an animation. The actual refresh of the screen column by column was done by timed interrupt, so it remained consistent no matter what else was going on on the PIC.

We got three pixel levels (ON, HALF-ON, OFF) by using two alternated screen buffers. A pixel that was half brightness was on in one buffer and off in the other. That way it got half duty cycle. (Actually it only got 1/3 duty cycle because we displayed the second buffer two times out of three. That was because it made the contrast between all-on and half-on better.)

11.4 Precedents

Precedents for *Hotpants* are somewhat hard to find. It seems that existing technologies are always either more or less than *Hotpants*. Handheld displays that do more than *Hotpants/LittleVision* are everywhere. These are on PDAs and the backs of digital cameras. There are beginning to be backlit

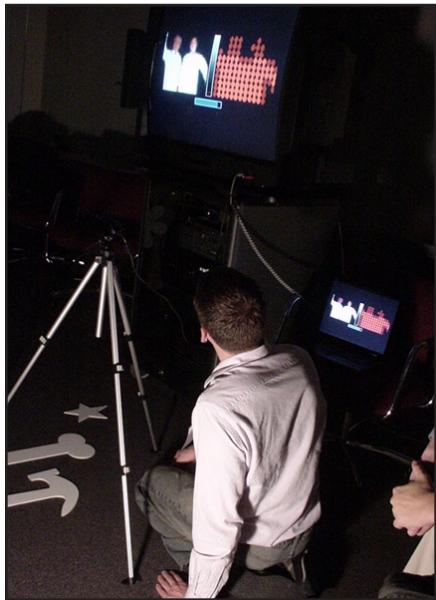


Figure 11.5. Justin filming a tiny movie.

LCD picture frames sold, which are somewhat similar in spirit to *Hotpants*, but deliver more image fidelity than object-relationship. Products less than *Hotpants* are the LED array components themselves, which come in a huge variety of sizes and configurations but have no built-in control circuitry to drive them.

Pixilated LED displays are everywhere as banners, and even architectural surfaces. People are starting to have video displays as small as watches. But all of these try for an imagistic resolution. An exception is Jim Campbell, an artist whose work with LED arrays explores pixilation, motion, blur, and form. His pieces led me to realize that putting a blurring filter over a highly pixilated display makes the image easier to decipher. His pieces also demonstrate how much recognition we get from motion.

11.5 Evaluation and Critique

Hotpants/LittleVision brought my attention to the realm of the handheld object, a scale which allows users to form totally different kinds of attachments than room-sized environments. *LittleVision* essentially compressed room-scale activity and placed it in the hand as a small electronic brick with a pleasant heft. Participants had a connection with the scenes they were filming, and then immediately thereafter held them in their palms. It was a very different experience than it would have been to see them on a television screen, or even on the LCD panel of a handheld video camera. This difference had a lot to do with a level of abstraction that the limited resolution enforced.

10 X 14 is not very many picture elements. Complex scenes are not recognizable. This forced an act of imagination onto the experience of viewing a *LittleVision* that, like the cartoon rendering



Figure 11.6. A digital picture frame from Ceiva. [<http://www.ceiva.com/>]



Figure 11.7. The Nasdaq exchange in New York has a full color LED wall.

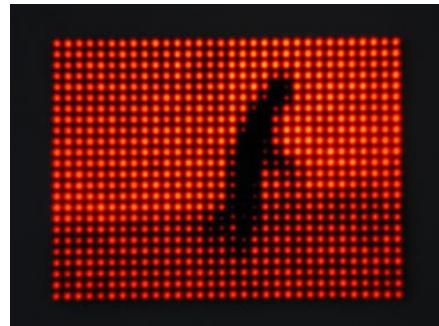


Figure 11.8. From Motion and Rest #5, Jim Campbell, 2002. [<http://www.jimcampbell.tv/>]

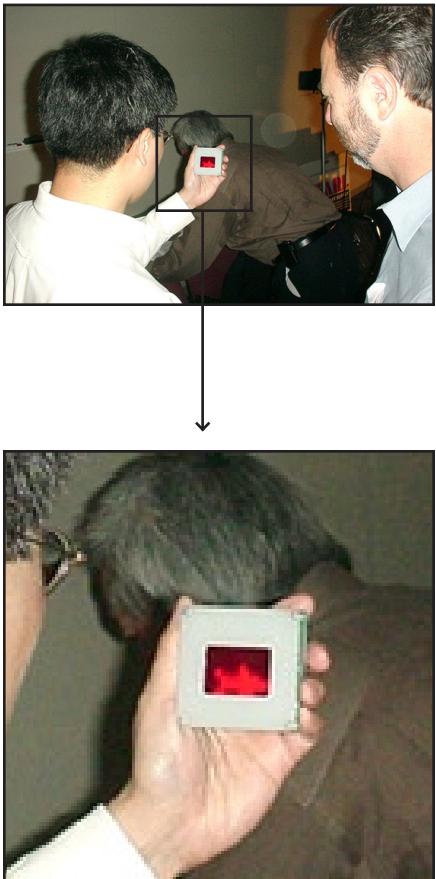


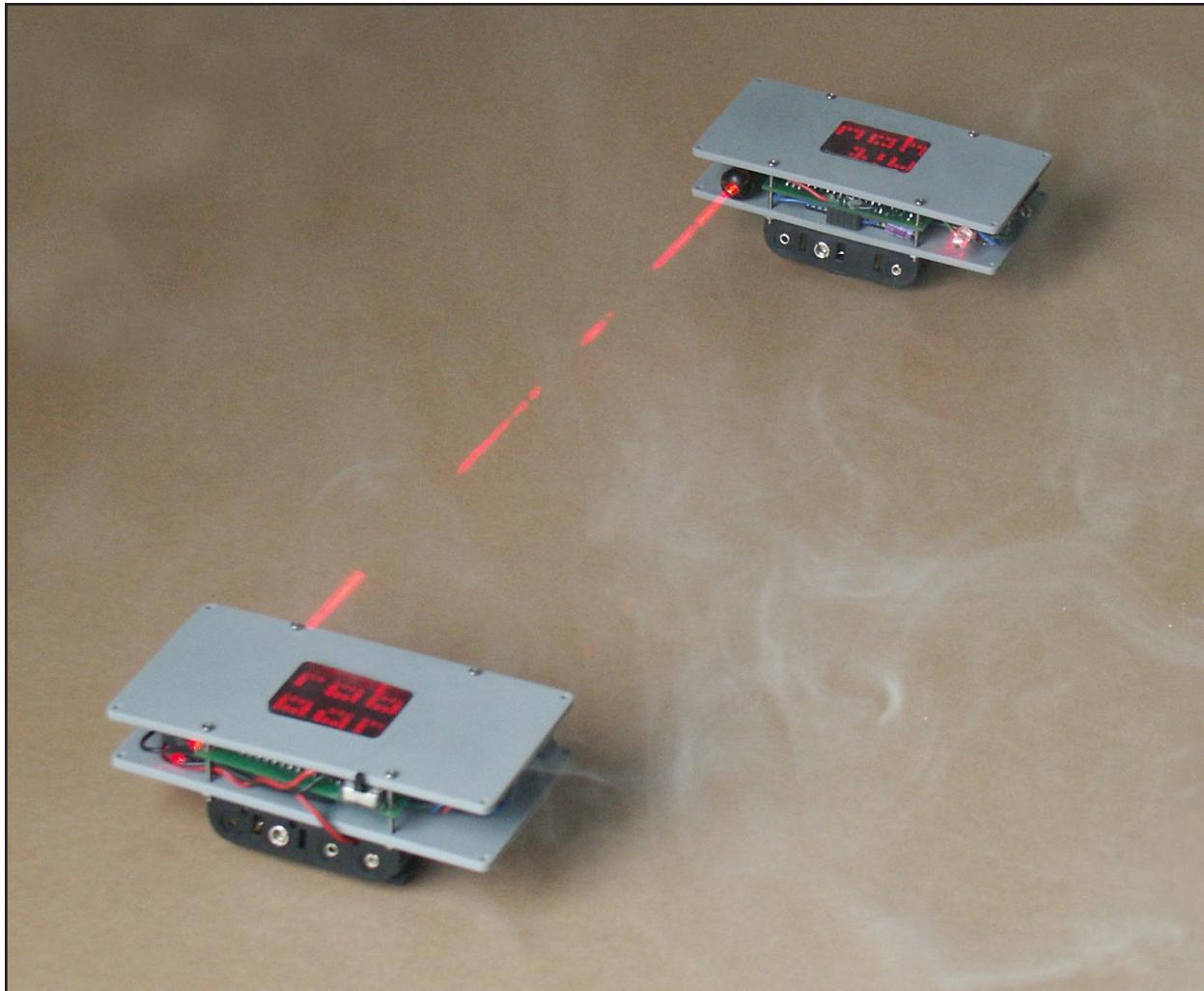
Figure 11.9. Can you tell what this movie is about? (Hint: It swims in the ocean and has big sharp teeth.)

discussed above, removed the distracting quality of near-perfection. The viewer could slip in and out of seeing figure, ground, or even individual pixels. This slippage was also tied tightly to the distance at which the object was viewed, which made people experiment with it, bringing it close to their faces or holding it as far away as possible. As in Campbell's work, scenes that were impossible to understand would sometimes snap into focus when they started to move. Interestingly in *Installation* it was also motion that brought out the sharpest qualities of depth. Human visual recognition owes a great deal to motion [Nakayama, 1985].

The screens on PDAs are designed not to confuse the issue of pixel versus image. They display images as faithfully as they are able at high enough resolution that they are instantly recognizable. Their displays are primarily surfaces of interface, which take up as much of one side as possible. The interface draws the user's attention to a flat space of text and buttons, which totally overpowers the substance of the object itself. They are always fighting their physical existence, trying to become thinner and lighter. They are rectangular to overlap the palm in one dimension. This makes it impossible to fold one's thumb down across the top of them—the natural desire for holding palm-sized objects. They are held like a stick, not a rock. There is something that destroys intimacy having to wield an object that you would cup if you could.

The object-qualities of *LittleVision* are its most important. It is appealing because its size, shape, and weight make sense to the eye and hand. Any smaller, and it would fall below a satisfying palmful. Any larger and it would be unwieldy. *LittleVision* satisfies my goals for spatial computing by its being a computational object that compresses space into a small body while not having its object nature over-powered by its computational process.

12. Pointable Computing



12.1 Introduction

One way to understand remote communication is as a battle with the separating qualities of space. AT&T's old slogan "Reach out and touch someone," made that explicit. The phone was to be an electronic prosthesis for contact. But it has not only been long distances that we have put effort into nullifying. The "remote" in remote control typically connotes no more than 15 feet. This kind of spatial collapse attempts to bring things just out of the sphere of reach into contact with the fingers. It functions as an extension of touch, and most

Figure 12.1. Word Toss handhelds sending information over a visible laser.

remote controls resemble the kinds of interface we would expect to encounter on an appliance itself. This is not an interaction about communication, however. It is strictly about control, and it operates unidirectionally.

Remote control has moved a step further in recent years to encompass remote data access. This has pushed the technology beyond the capacity of infra-red communication and into radio-frequency territory with 802.11 and BlueTooth. The spatial idea behind these technologies is different from the directed-connection spatial model of telecommunication and remote control. Instead, these technologies are proposed to replace wires. Wires are simply physical connectors designed to carry signals. They do exactly what their shape implies. It has been possible until recently to tell what a machine is connected to by tracing its wires. Now the wires are going away, and it is totally unclear what connections are being made from machine to machine. A useful assumption may be that everything is connected to everything. There is no disconnect to privilege any one particular connection over another.

And that is a problem. Now that we have essentially conquered spatiality with communication technology, we are left floating in an undifferentiated spacelessness. True we may have eliminated the need to crawl around to the back of our machines to plug in devices, but we have replaced that inconvenience with a new burden of reference. We must assign everything with which we want to communicate a unique identifier so that we can select it from a list of things in range of communication. We have essentially become like our machines, who have no notion of directionality or focus, and therefore must refer to things by ID. This is not a humanizing direction of development.

What I proposed in *Pointable Computing* was a solution to this crisis of non-space in wireless communication.

12.2 System Description

Pointable Computing was simply a handheld system for remote communication over visible lasers. It was the absolute epitome of directed communication. Until I learned to spread the beam slightly, it was so sharply directed that it was hard to use at all. The purpose of the project was to explore the possibilities and experiential qualities of highly-directed communication and contrast it with undirected technologies.

Technical description

The system consisted of two handheld devices equipped with laser-diodes and phototransistors for sending and receiving of signals. I spread the beam slightly with a lens system to make it eye-safe and easier to control for distant targets. Each handheld had a display board (a repurposed *Hotpants* display), a single button and a control wheel. I also made a standalone wall-mounted receiver with three *Hotpants* displays. Each of these systems was driven by a PIC microcontroller.

Word Toss

The proof-of-concept application I designed for the devices I called *Word Toss*. Each handheld showed two words stacked vertically, a transitive verb on top and a noun on the bottom. In one of the devices, rolling its wheel changed the verb, and in the other device, it changed the noun. Each device's laser was on by default. When the devices were aligned, their lasers hit the other's receiver, and a pixel in the top right of the receiving device would light to indicate that it had acquired a signal. When

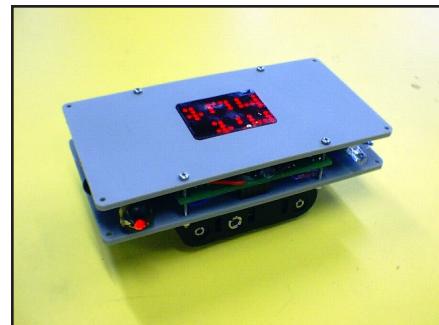


Figure 12.2. One of the two handheld devices.

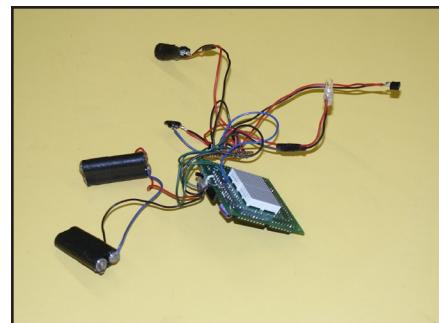


Figure 12.3. The innards of the handheld. Clockwise from top: laser, send button, receiver, *Hotpants* display, batteries, more batteries, and thumbwheel.



Figure 12.4. It was also possible to use the handhelds to send messages to this wall-mounted unit.

either device's button was pressed, its laser was pulse-modulated to send a message to the other device. The message in *Word Toss* was simply the verb or noun selected with the wheel. The other device received the message and changed its word to match the word sent. It was also possible to use the handhelds to send words to the wall-mounted device, which displayed them. I was successful in sending messages from more than 30 feet away.

12.3 Precedents

Sending information on visible lasers has been done for a long time. The field is called “Free Space Optics” to denote that it is an optical transmission that requires no medium of conveyance other than air. Systems have been developed that can communicate up to 2 miles and at speeds of up to 622 Mbits/sec. These devices do not use visible lasers, and are certainly not handheld. But they do demonstrate a kind of spatial pointing that is similar in spirit if not in scale to *Pointable Computing*’s.

Certainly existing infra-red communication, which tends to have a conical spread, was a datum. The most common specification IrDA allows many devices to communicate with each other. The IrDA specification demonstrates a vision very much like the point-and-communicate model of *Pointable Computing* [IrDA, 2000]. The deficiencies of infra-red LEDs are that their illumination quickly spreads too much to be detected, making it suitable only for very short range applications. By contrast low power visible lasers can extend hundreds of feet without dissipating too much to be useful. The limiting factor at long range with handheld lasers is the unsteadiness of the hand, which distance multiplies. The other liability of infra-red is that

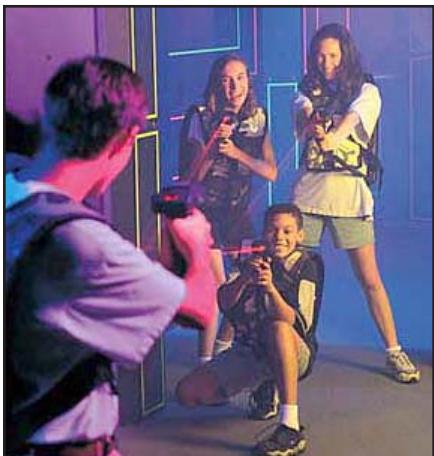


Figure 12.5. Laser tag systems have used the total directedness of visible lasers for entertainment. [<http://www.lazerrunner.com/primer.html>]

it is invisible, so the user is never quite sure if a connection is being made or is reliable.

Pointable Computing's second lineage is the whole family of computational pointing input devices.

Figure 12.6.
Pointing Devices Organized by Domain of Operation

Metaphorical

Arrow Keys



Joystick



Trackball



Mouse



Tablet



Documentary

Camera



Range Finder

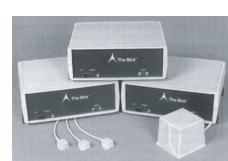


3D Digitizer



Literal

Inductive Sensors



Eye Tracker



Touch Screen



Pointable



Metaphorical devices map spatial gestures and actions to a metaphorical machine space.
Documentary devices recover spatial qualities of the world and report them to the machine.
Literal devices are active operators that report an actual spatial or bodily condition. Literal devices differ from metaphorical devices in that they directly connect agent, environment, and machine. Metaphorical devices are concerned only with the input environments locally relative to themselves.

12.4 Use Scenarios

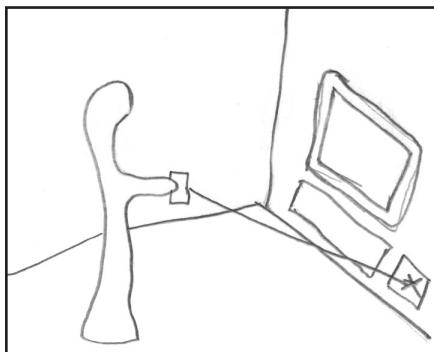


Figure 12.7. Universal remote sketch.

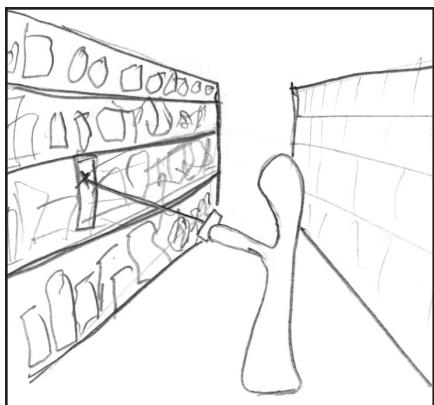


Figure 12.8. Active tagging sketch.

I developed several use scenarios to illustrate possible applications of pointable computing.

Universal remote

The most obvious use of *Pointable Computing* would be to make a universal remote. Pointing the device at any enabled object would turn the handheld into a control for that object.

Active Tagging

Imagine yourself walking down an aisle of products. You see one you would like more information about or two you would like to compare. You point your handheld device at them and they transmit information about themselves back to you. Why is this different from giving each product a passive tag and letting an active reader look up information in a database? The answer is about autonomy and decentralization. If the information is being actively sent by the object scanned, it does not need to be registered with any central authority. It means that no central server of product information is required, and anyone can create an active tag for anything without registering some unique identifier. Note also that in this scenario we see the likely condition that a non-directed wireless communication like BlueTooth would be useful in conjunction with a Pointable. The two technologies complement each other beautifully.

Getting and Putting

In a vein similar to Brygg Ullmer's *mediaBlocks* project, it would make sense to use *Pointable Computing* to suck media content from one source and deliver it to another [Ullmer, 1998]. Here again it would not be necessary to display much on the handheld device, and one button might be sufficient. An advantage in media editing that

the Pointable has over a block is that there is no need to touch the source. That means that it would be possible to sit in front of a large bank of monitors and control and edit to and from each one without moving. It might even make sense to use a Pointable interface to interact with several ongoing processes displayed on the same screen.

Instant Wiring

In this simple application, the Pointable is used simply to connect together or separate wireless devices. If, for instance, a user had a set of wireless headphones which could be playing sound from any one of a number of sources, there is no reason they couldn't simply point at the headphones and then point at the source to which they wanted it connected.

Sun Microsystems likes to say, "The network is the computer." This is a fairly easy formulation to agree with considering how many of our daily computational interactions are distributed among multiple machines. Any form of electronic communication necessarily involves a network. The shrinking and embedding of computation into everyday objects implies that informal networks are being created in the physical fabric of our homes and offices. If we assume that the network of wireless devices around ourselves is essentially a computer, we must admit that we spend our days physically located inside our computers. Being located inside the machine is a new condition for the human user, and it allows the possibility of directing computation from within. A pointing agent, a kind of internal traffic router, is one potential role for the embedded human being.

Reactive surfaces

Reactive surfaces are building surfaces, exterior or interior, covered with changeable materials

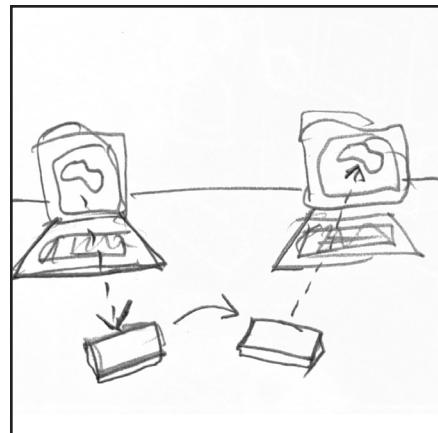


Figure 12.8. Getting and putting sketch. The pointable sucks up information from the computer on the left and spits it out onto the computer on the right.

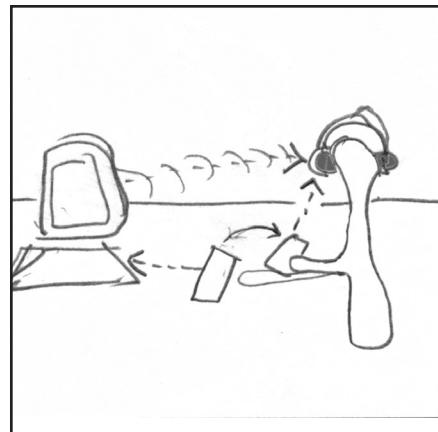


Figure 12.10. Instant wiring sketch. The user connects the computer to headphones by pointing at it and then at them.

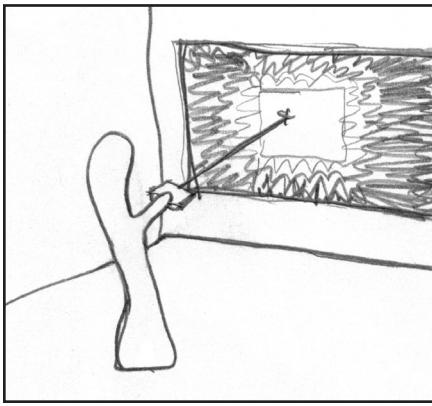


Figure 12.11. Reactive surfaces sketch.
A user makes a patch of transparency in
an opaque surface.

coupled to arrays of pointable sensors. They make use of new materials that have changeable physical properties such as LCD panels, electrochromic glass, OLEDs, or electroluminescents. It would be possible with a pointable device to write a temporary message on a desk or wall or define a transparent aperture in an otherwise shaded window wall. Such an aperture might follow the path of the sun during the day.

12.5 Evaluation and Critique

Pointable Computing takes as its starting point an emerging reality in which everyday electronic devices communicate wirelessly. These devices already have identities tied to their functions, be they headphones, storage devices, or building controls. They are not crying out for an additional layer of interface. How can we address the new capacity of things to talk to each other without further mediating our relationships with them? We need the remote equivalent of touch, an interaction focused on its object and containing its own confirmation. *Pointable Computing* offers that by way of a visible marker, a bright spot of light. The user does not need to consult a screen to determine if they are properly aligned. It is apparent. The receiver may indicate that it has acquired the beam, but that indication will always be secondary to the visual confirmation that the object is illuminated.

The system did feel substantively different from existing modes of wireless communication. And its primary difference was its spatial specificity. It felt much like using a laser pointer, which has a remarkable quality of simultaneous immediacy and distance. This I believe is due to its antiphysical quality of tremendous length with infinite straightness and lightness. It is like an ideal rod. Also like a physical pointer, *Pointable Computing*

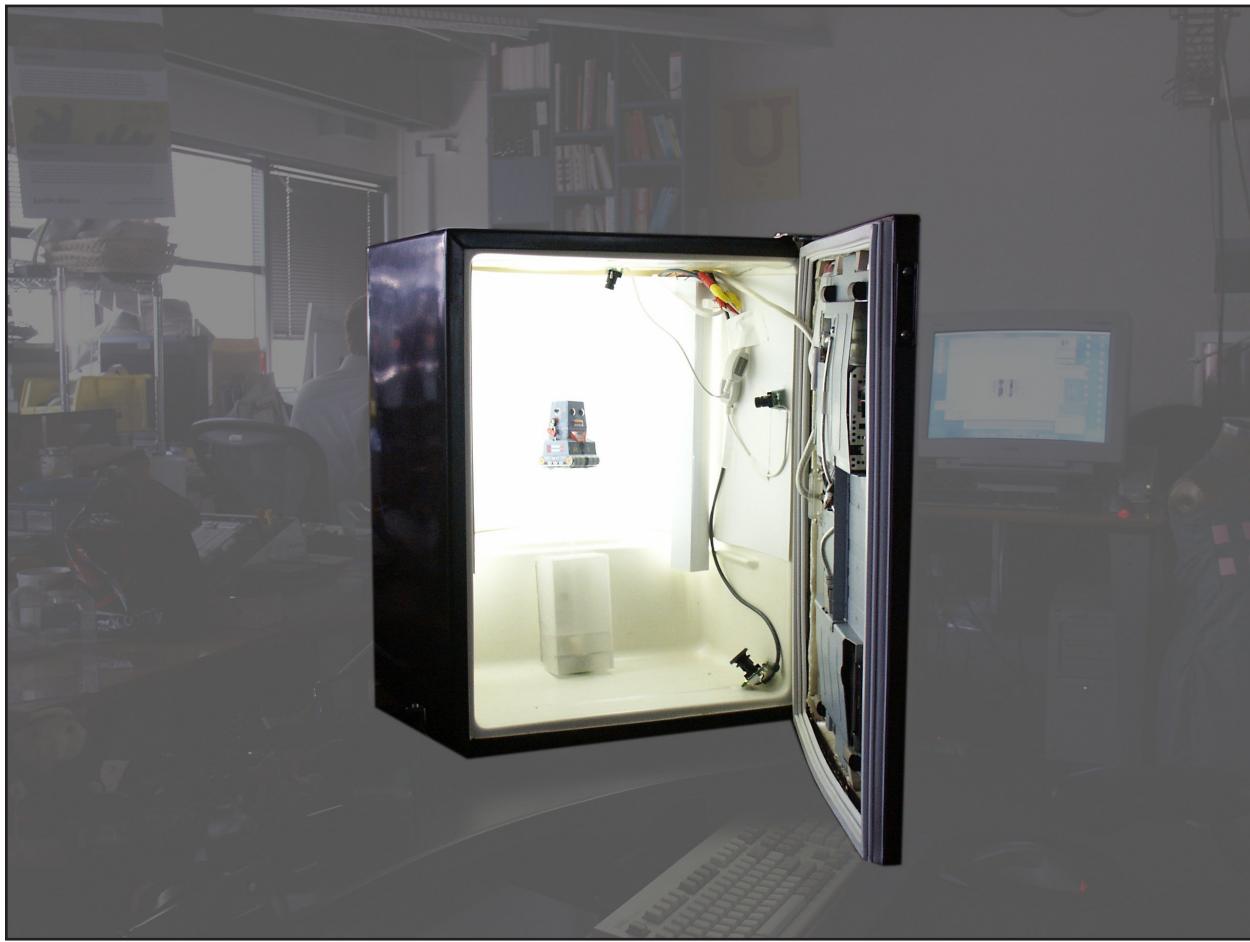
is usable because it offers feedback. As can be demonstrated by a game of “pin-the-tail-on-the-donkey” we cannot point very well without continuing to reference what we are pointing at. A laser spot is the perfect feedback for pointing—ask a sniper with a laser scope.

As Norbert Weiner pointed out, any system containing a human being is a feedback system [Weiner, 1948]. As users, people automatically adjust their behavior based on the overall performance of the system. What makes the *Pointable Computing* a robust communication system is that the feedback loop containing the human being is direct and familiar. The human eye has an area of acuity of 1–2°, implying that narrow, beam-like focus is the norm, not the exception for human perception. The rest of the visual field is sampled by eye movements and then constructed in the brain. Tight visual focus is the way we solve the problem of reference without naming in a spatial environment. The feedback loop that enables the act of looking entails our observing the world and correcting our body attitude to minimize error of focus. It happens so quickly and effectively that we do not even notice it. The same feedback loop can be applied to a point of focus controlled by the hands. It is not quite as immediate as the eyes, but it is close. And, as it turns out, it doesn’t suffer from the kinds of involuntary movements that plague eye-tracking systems. This is the same mapping as the “eye-in-hand” virtual window model.

Pointing is an extension of the human capacity to focus attention. It establishes a spatial axis relative to an agent, unambiguously identifying anything in line-of-sight without a need to name it. This brings our interactions with electronic devices closer to our interactions with physical objects, which we name only when we have to.

Pointable Computing successfully takes computation away from the screen and into the space between things. Its use of simple, inexpensive components, and its surreptitious hijacking of the human machine as a very fine controller make it more appealing than many other options like motion-tracking, inductive position sensing, or computer vision for establishing simple spatial relations to a user. It requires no calibration, it operates robustly under almost any conditions, and it weighs next to nothing. I expect to see more systems employing laser-directed spatial interaction.

13. EyeBox



13.1 Introduction

My final project at ACG turned my attention very much toward physical objects. It focused on finding a good way to get them into the computer. As many people, including Bill Buxton have noted, even as our machines get tremendously more powerful internally, our abilities to get things other than printed material in and out of them have not progressed very far. The engines of computation have digested very little of our world. In order for our machines to become fuller partners in our work and play, they are going to have to join us in

Figure 13.1. The *EyeBox* 3D scanner with a small windup robot inside.

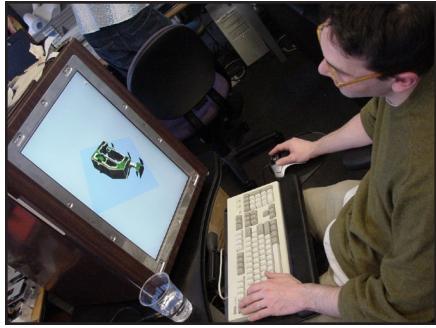


Figure 13.2. Working with a scanned form (a vice) on the computer.

our physical world. That means we are going to have to introduce them to the objects that form the substance of our lives.

In *EyeBox*, I made a computer at which a user could sit and work, and then they could open it up and place an object inside. The object was scanned in 3D and its form became available for digital manipulation. However important it was as an inexpensive 3D scanner, I think it was more important as an example of a simple spatial interaction with a computer that made sense to everyone who tried it. How should we get an object into the computer? Open it up and put it in. That is how we get anything into anything in the real world. It broke the barrier of the screen by making it a door that hinged open to reveal a usable space behind it.

13.2 System Description



Figure 13.3. The screen was embedded in the door of a mini-fridge.

EyeBox was made out of mini-fridge, three webcams, two fluorescent lights, a microwave turntable, and a flat panel display. Any dark-colored object nine inches on a side or less could be placed into the box, and in approximately twenty seconds, the machine rotated the object once around and produced a full volumetric reconstruction of it from the visual hull of 24 silhouette images (eight from each camera taken during the rotation). The user began by opening up the fridge. They would place an object on the turntable inside, which had hash marks around its edge. They would close the fridge, and the turntable would begin to spin. The user would see the camera images from the three cameras displayed onscreen as the object rotated. After a full rotation, the screen would change to a 3D projection showing the 24 silhouette images in their positions around the platform, and an iteratively refining 3D reconstruction of the object

on the platform. Over the course of the next few minutes, the representation of the volume of the object would get progressively finer until it reached a resolution of 512 X 512 X 512 voxels. Then it was filtered to smooth the voxels, giving it a smoother shape.

13.3 Motivation

A goal in the project was to keep costs low. Very nice 3D laser digitizers are available for \$8,000. *EyeBox* is not as accurate as these, but it cost \$100 to build (minus the flat panel, which is necessary to the spatial interaction but auxiliary to the scanning). There is an obvious place for such inexpensive devices in industries such as rapid fabrication, design, and entertainment.

Less obvious, but perhaps more important in the long term is the need for computers to be able to recover geometries from the world simply to be more useful in problems that are meaningful to human beings. Computers are wonderful devices for cataloging objects. It would be great to be able to catalog objects as full three-dimensional reconstructions of themselves. These representations could be sent to others and printed out either locally or remotely, yielding respectively a 3D copier, and a form teleporter. Museums might be interested in this to catalog artifacts or to exhibit pieces in a way that users could place them in a cabinet to find out more about them. It could be used to let people leave impressions of objects in places where they would not leave the actual object.

13.4 Technical Description

EyeBox used a technique called *visual hull reconstruction* to recover volumes from the silhouettes of objects. Methods of visual hull

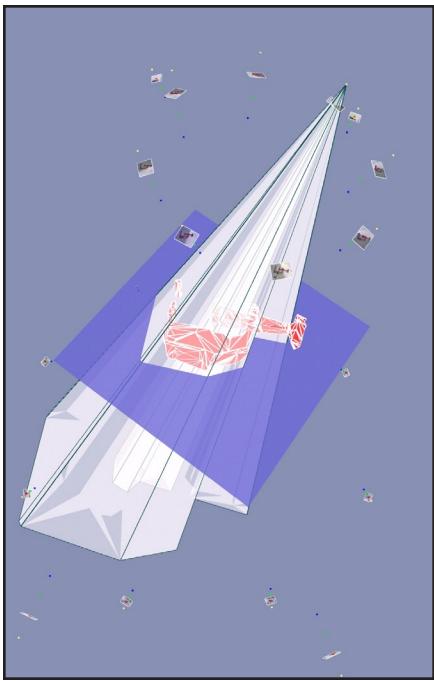


Figure 13.4. The silhouette of the vice relative to the camera on top represents a cone of volume in which the vice must lie. The intersection of this cone from each camera is the visual hull.

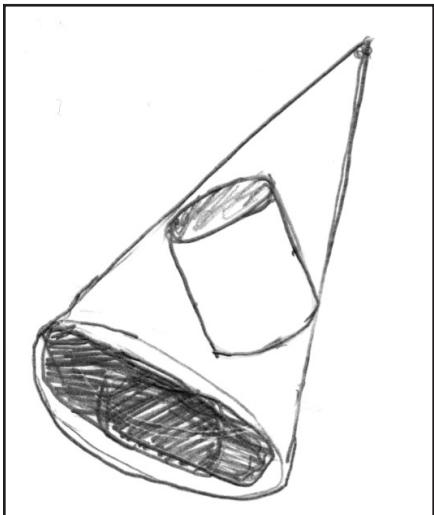


Figure 13.5. Figure showing how a cup's rim always hides its inside in silhouette.

processing fall loosely into three categories: image-based [Matusik, 2000], polyhedral [Matusik 2001], and volume carving [Szeliski, 1993]. All of these techniques rely on the same basic principle—that a silhouette relative to a calibrated camera produces a generalized cone of volume in which the object must be located. These cones from several cameras can be intersected to produce a representation of the volume at which they are all looking. It takes surprisingly few cameras to get a fairly good approximation of most common shapes.

Techniques for reconstructing form from silhouette data are all capable of producing its “visual hull” relative to the views taken. Abstractly, the visual hull of an object is the best reconstruction that can be made of it assuming views from every angle. The visual hull, as discussed in Petitjean, is a subset of an object’s convex hull and a superset of its actual volume envelope [Petitjean, 1998]. Specifically, a visual hull technique cannot ever recover a full topographical concavity, such as the inside of a bowl. Such an indentation will be filled in by the visual hull. This is because the technique reconstructs volumes from their silhouettes, and no matter what angle one views an object from, a complete concavity will be obscured by its rim in silhouette.

Image-based

Image-based techniques are the fastest because they do not reconstruct three-dimensional form at all. Instead they synthesize new views from any angle by selectively sampling from the source images directly. Since there is no volumetric representation produced, they are not suitable for true volumetric reconstruction problems. It is possible to imagine, however, reformulating many volumetric problems as image-based problems. For instance, volumetric object-matching may

be construed as an image search for the best reconstruction to match a given image of an unknown object. The challenge would be making it fast enough to search all possible orientations of all possible matching objects.

Polyhedral

Polyhedral techniques produce a surface representation of the object (easily converted into a volumetric representation if required) by geometrically intersecting polygonalized versions of the cones. This is relatively quick, and provides an unaliased representation without the need for iterative refinement. Extensions to this technique are able deal with the hulls as splines to let them curve as in Sullivan and Ponce [Sullivan, 1998]. Polyhedral techniques allow for easy texture-mapping of the original images back onto the reconstructed surfaces, giving another level of detail. I implemented this technique in several different ways, but each time I ran into the same problem: it is highly sensitive to calibration and numerical error. It is imperative that the geometric operations used to construct the volumes be numerically robust and have adjustable geometric tolerances. Methods for general volumetric intersection (constructive solid geometry) that have these necessary characteristics are challenging to implement and difficult to find as free software libraries. So although in theory this may be the best class of methods, it is very difficult to get it to work reliably on real-world data.

Volume carving

This is the simplest technique to implement and also the slowest. It projects voxels from world space onto each of the camera views. If a voxel projection falls fully outside any of the silhouettes, it can be discarded. This produces an explicit volumetric representation at the cost of voxel aliasing and lots

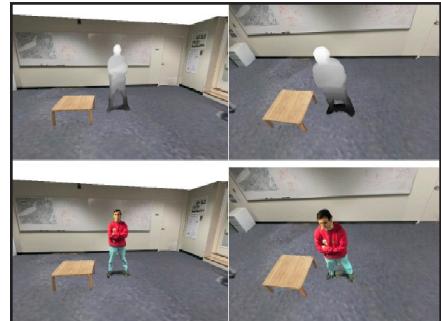


Figure 13.6. Image-based visual hulls from [Matusik, 2000].

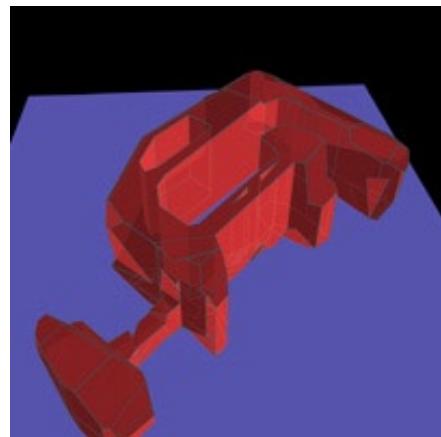


Figure 13.7. A polyhedral reconstruction of a vice produced by the first version of *EyeBox*.

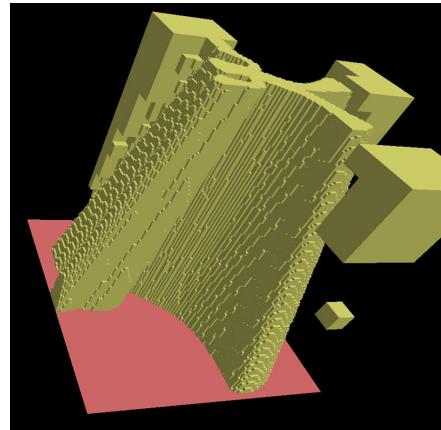


Figure 13.8. Gumby as volume-carved by a single camera from the top.

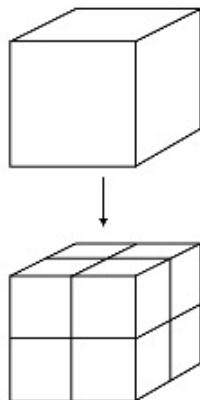


Figure 13.9. An octree node may be subdivided into eight subnodes.

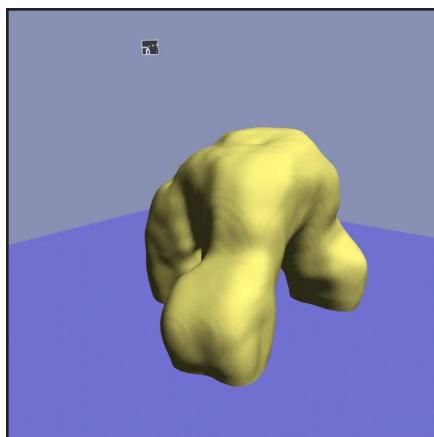


Figure 13.10. An object after smoothing.



Figure 13.11. A swept-laser 3D digitizer from Minolta.

of computation. I implemented it because I wanted a volumetric representation for matching purposes and it was the easiest to produce. Because of its aliasing it is also somewhat more tolerant of error in camera calibration than the polyhedral method. This proved to be a significant advantage in the turntable driven scanner.

Octree subdivision

Having chosen the volume carving method, I sped it up by representing the volume as an octree. That is a recursively refined volumetric tree starting with a root node representing the entire volume to be scanned. When a projected node was found to be cut by the silhouette from any camera, it was divided into eight subnodes. This way whenever a large node was found to be outside of any of the projections, it needed never to be subdivided or otherwise considered again. This sped processing up dramatically. Another speed advance was to iteratively refine the octree representation by one level at a time, running it on each camera at each level. That way more large octree nodes were rejected earlier, and did not slow it down. Octree nodes that were wholly inside each silhouette were marked too, so that on each iteration, the only nodes that had to be processed were nodes that in the previous level intersected silhouette boundaries in some camera. This was tantamount to finding the substantial structures early and then iteratively refining their surfaces. It also meant that the user saw the form improving over time and was free to stop the process whenever it got to a level of detail they were happy with. I smoothed the surface by applying a Gaussian filter to the voxel data and then finding an isocontour.

13.5 Precedents

There are existing 3D digitizers which work more accurately than *EyeBox*. One such family measures

parallax in a laser beam that it sweeps over the surface of an object. These return point clouds which can then be surfaced. They are, however, two orders of magnitude more expensive than *EyeBox*.

The technique of reconstructing volume from silhouette data is not new. It is well worked out and documented in a variety of sources. But it is surprisingly rarely implemented. Perhaps that is because of a fear of incompleteness—if it can't promise to accurately digitize an arbitrary form, what good can it be? I propose that a rough scan can be quite useful. Typical setups for the process involve a single well-calibrated camera viewing an object on a turntable as in Kuzu and Rodehorst [Kuzu, 2002]. The turntable is turned by hand or motorized to provide an arbitrarily large number of silhouette images to be acquired from a single camera.

Fixed multiple camera setups exist, notably Matusik, Buehler, and McMillan's [Matusik, 2001], which is capable of scanning people in a room in real time. This setup requires a computer per camera and one more as a central processor, so it doesn't qualify as a low-cost solution, but their results are stunning. It is also not designed for scanning palm-sized objects.

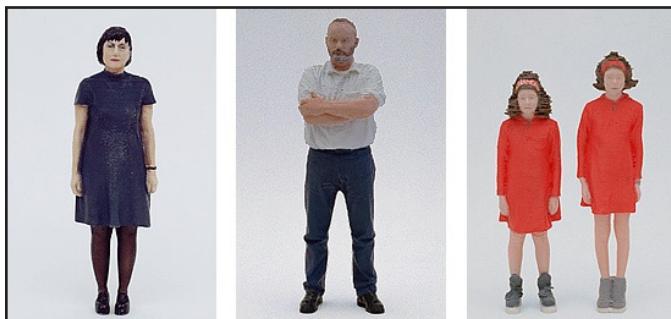


Figure 13.12. Four 1/10th life-size people from *54 Personen 1:10*, 1998-2001, by Karin Sander. [<http://www.karinsander.de>]

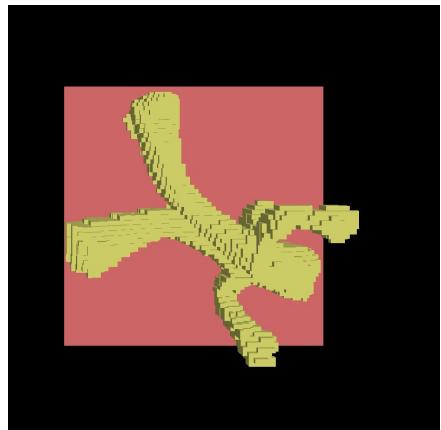
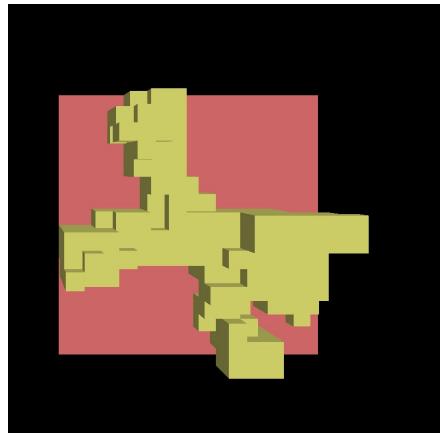
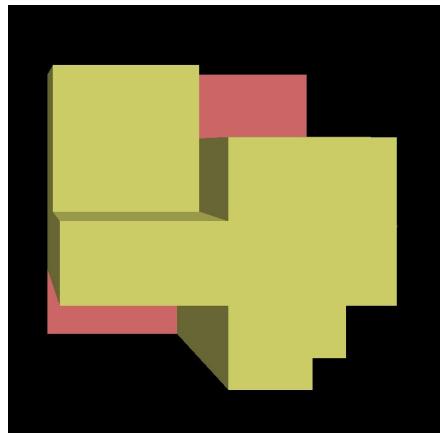


Figure 13.13. The iterative refinement of Gumby. The user can stop the process at any time.

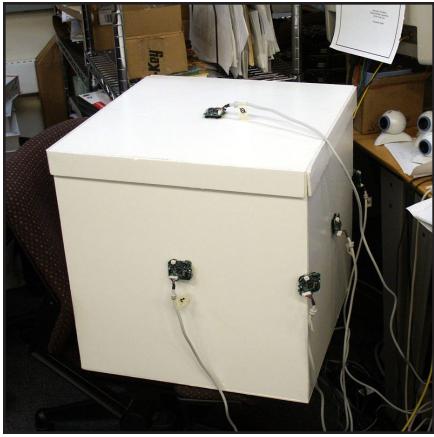


Figure 13.14. The first version was housed in a foamcore box.

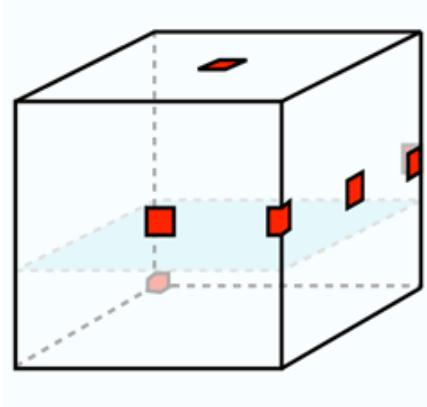


Figure 13.15. The six cameras were disposed as shown.



Figure 13.16. With the lights on and the original calibration cube inside .

There are also cultural precedents related to 3D scanning. Karin Sander took full body scans of 54 people and then mechanically reproduced them at 1/10th life size. She then painted them from photographs and displayed them on pedestals.

13.6 Design and Operation

EyeBox as a mini-fridge is a second generation of the system. The first was less refined.

Version 1

The first version, a foamcore cube 18 inches on a side with six cameras at fixed locations and no turntable, was quite successful. The camera positions in the original version had to be carefully chosen to deliver the most amount of non-redundant information. Therefore they were not one-to-a-side, as might be supposed. Views separated by close to 180° are primarily redundant for silhouettes.

The first step in the construction was the dismemberment of the webcams. Then I built an 18" X 18" X 18" cube out of foamcore and put a plexiglass shelf in it 7" from the bottom. I cut holes in the sides and top for the cameras and attached two small fluorescent lights to the inside. I then calibrated the cameras by Tsai's method embedded in a custom application I wrote for the system [Tsai, 1987]. Then I was ready to write the reconstruction software.

The first step was to acquire a silhouette image from each camera, which was very easy because of the well-controlled imaging environment. For each camera, I simply subtracted an image of the empty box and then thresholded the results.

The reconstruction proceeded as detailed in the octree method outlined above.

Problems

There were some problems with the reconstructed objects. Many of them had to do with the white background. Light colored objects did not scan well at all. Specularities on objects are always white and tended to be seen as background, drilling holes in objects. In a future version of the system, I would use a blue background to make segmentation simpler. Reflections off the plexiglass were troublesome. Finally, the box was rather large for an effective scanning volume of 6" X 6" X 6". That could have been improved with wider angle lenses, but the wider the field of view, the lower the quality of the reconstruction. There were also errors of volume just due to spaces not visible to any camera. This could have been helped with more cameras.

The second version of the system set out to solve some of these problems. It used a rotating platter to effectively multiply the viewpoints from three cameras into 24. The rotating platform also helped shrink the necessary size of the system. Since cameras were only looking at the object from one side, it was the only side that needed visual clearance. It imaged against a rounded background to get rid of dark corners in the empty volume.

Version 2

Version 2 was housed in a mini-fridge. I chose a mini-fridge because it fairly closely matched the dimensions I determined were optimal, and I could not resist the feeling of the seal made by a fridge door. I gutted the fridge and drilled a hole in the back to run cables out. I decided to orient it standing up rather than lying down so as not to evoke a coffin. Instead it is very clearly a mini-

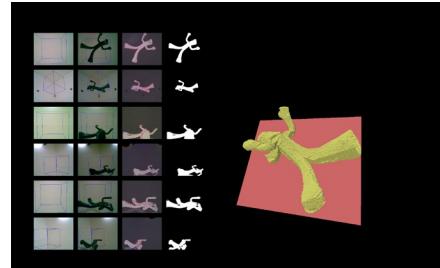


Figure 13.17. Gumby reconstructed. The images on the left show the silhouetting process.

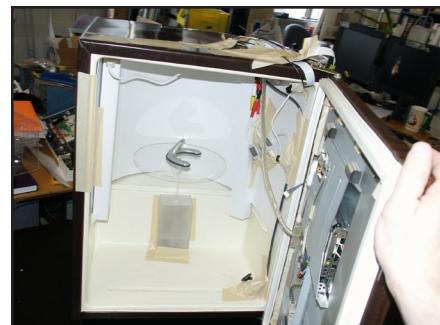


Figure 13.18. Version 2 under construction.

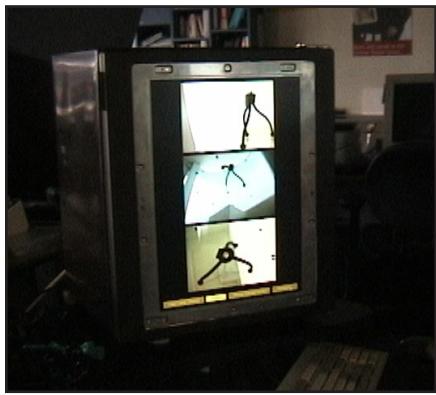


Figure 13.19. The three cameras show real-time feeds on the display.

fridge, and its hybridity is part of its strong appeal. I used a water-jet cutter to cut out a large opening in the door and mounted an Apple Cinema Display in it. I salvaged an AC gearhead motor from a old microwave turntable and mounted it inside the fridge with a shaft and a plexiglass turntable on it. I glued three webcams to the interior of the fridge looking slightly off-center at the turntable. I turned them off-center to maximize the probability that they would perceive the edges of objects—the source of all of my information. I was not concerned that they might not be able to see both edges at once because I rotated every object a full 360° . I disassembled two small fluorescent lights and mounted them inside the cabinet pointing directly back onto the curved white back surface. My hope was that this would completely backlight the subject and get rid of all the problems with specularity. In fact it ended up still giving a strong side light. I mounted a reed switch on the door hinge to control the platter motor. When the door closed, the platter spun.

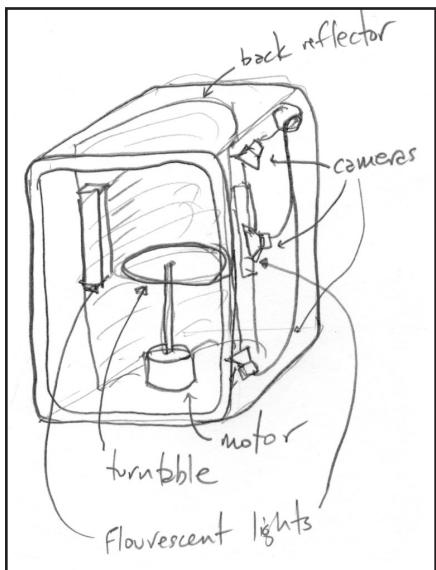


Figure 13.20. System diagram.

My setup avoided having to carefully control the speed or position of the turntable by placing black marks at its edges in 45° increments. The total light value from a small patch of the camera looking from the top was used to determine when the turntable was in position to use a single video frame from each camera as a still image from one angle. Two of the marks were not black—one was red, and one was cyan. These were present to indicate the starting position (considered zero degrees), and the direction the platform was spinning. It was necessary to determine the direction in real time because the turntable motor was a cheap AC motor lifted from a microwave, and it was therefore impossible to know which direction it would turn when power was applied.

I calibrated the cameras by the same procedure as the first version. Because I had not constructed the whole system to engineering tolerances, I calibrated each of the 24 views by hand rather than calibrating three and performing rotations on them.

Results

All of the changes proved to be advantageous, and my results were somewhat better with the new system. The biggest disappointment was how little it improved. The fantastic advantage of the technique is that it takes so little information to give very good results. After the first several cameras, adding more gives diminishing returns. It may be that 24 views is more than is necessary, and rotating the object may therefore be as well. With the current cost of webcams at about \$15, maybe I should just settle for 12 in a stationary setup. Not rotating has several advantages—easier, more consistent calibration, no moving parts, faster operation. The primary advantage, though, to not rotating the object is the improved magical quality of producing a transformable 3D reconstruction from an object that is totally stationary.

13.7 Analysis and Critique

EyeBox as a tool is more interesting as part of a workflow that passes objects between physicality and virtuality with ease. What it needs in order to do this is a matching rapid output device. The dominant model for architectural sketching is with layers of trace paper placed down over the top of an existing sketch. The sketch evolves as the layers pile up. A means to rapidly reproduce form could introduce this technique to three dimensional modeling.

Several people have pointed out that *EyeBox* would do very well coupled with *Installation*. Real objects

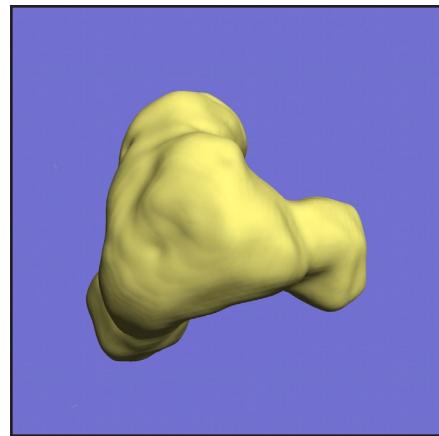


Figure 13.21. A scanned object. (A lump of modelling clay that looked a lot like this.)



Figure 13.22. A physical sculpture of a skull that has been warped by computational transformation and then rebuilt by hand in bone. [Robert Lazzarini, “skulls,” 2000]

could be virtually duplicated and placed in other parts of the space. Or an object that is needed as a reference could be duplicated and placed virtually in the room before the actual object is removed. I look forward to bringing to two together. It should not prove too difficult.

A question that *EyeBox* naturally raises is what is the relationship of a formal duplicate to the original. It is a simulacrum, but functional only in so much as the function of the original derives from its form. And as a purely digital representation it can serve no physical purpose without transduction back into the real world. The power of the digital model is its extreme malleability. We can operate on it in ways we could not imagine working on physical objects. We have used the idealized machine space to break physical law. Hybrid real/virtual form-making promises to push future design in new directions.

14. Summary Conclusions

In the course of building and analyzing six separate experiments in spatial computing, a few fundamental design principles have become apparent. These are not highly theoretical or comprehensive but do offer a set of guidelines for implementations of computational systems that engage real space. I have not done enough to put forward a comprehensive theory. My hope is that that will never be possible, allowing me to work productively toward it for the rest of my career. The variousness of the experiments has been essential to the inductive process. The qualities evident from experiments so widely disparate in scale and approach are likely to retain some validity over the entire field.

Rather than restate everything from the analyses of the individual experiments, here I will organize the results briefly by theme and then propose a vision of a future application for the field.

14.1 Editing

Certain decisions must always be made in how the real world will enter into the system and vice versa. These are essentially questions of editing.

Suggestion vs. realism

The power of the live video feed in *Installation* demonstrated that often a suggestion of the real is enough (particularly if the system is not immersive, so that the simultaneous experience of the real is not denied). The camera was not much to add, and it certainly did not fool the eye. But it established context of the interaction. The mind did the rest of the work.

This is closely related to the kind of abstraction at work in *LittleVision*, which presented a highly

pixilated, low-bandwidth representation of a visual scene. It was just enough to indicate what was going on. The work of the mind to bring the scene into focus created an attachment to the interaction. The understanding was a collaboration between the object and the observer.

The message is this: Do not try to fool the senses! It will not work. The senses are canny and aware of much more than we will ever be able to simulate. The closer we try to approximate reality, the more noticeable our failure will be. Instead, we must suggest what we want the experience to be and trust the mind of the user to do the rest.

Literalness vs. metaphor

Donald Norman in *The Invisible Computer* inveighs against metaphor, arguing that it always breaks down somewhere [Norman, 1998]. Since there is no true equivalence there must be difference that the user cannot know when to expect. This may be true, but I think a worse crime of metaphor is its obscuring the identity of something real. However, there is often no real referent for the metaphors of computer interface, making them essentially free-floating and instantly changeable. Hayles calls them “flickering signifiers” [Hayles, 1999; 31]. There is a fantastic convenience to working inside a metaphor factory that runs on nothing. There is no need for the cumbersome limitations and specificities of the real. This cannot be denied, and I do not think that all computer interface should give up metaphor. I believe that there is currently no better alternative for the control of most existing software. However, it turns the screen into an impenetrable barrier, and it has no place in spatial computing, which has as its fundamental objective the union of the real and computed. Objects should be themselves and should not have to be referenced by an icon or a name.

In graphical interfaces for all of these projects combined, I believe I used a total of three buttons and zero icons. I consider the buttons a mistake.

Design vs. engineering

The physical objects involved must be the products of design, which goes beyond a process of engineering. They should be approachable and pleasing. They should not deny their physicality by trying to disappear, but use their forms for all of their potential value.

There are many factors at work in whether an object will have resonance with an audience. I identify two.

First, if it is to be held, it must have a form that is pleasing in size, weight, and texture. *LittleVision* demonstrated this.

Second, an object needs its own identity and autonomy. It must have no wires trailing off it unless the wires are specific to its identity. Besides making an object a crippled dependent on something external, wires restrict its manipulation.

A helpful technique, not exclusive of the others, is to use objects with existing resonance and repurpose them. The mini-fridge cabinet of *EyeBox* gave it an appeal to many that no custom cabinet could have.

14.2 Qualities of interactions

One of the fields closely related to spatial computing calls itself *interaction design*. The field does not limit itself to interaction with computers, but extends to other products of design such as architecture. The notion that interaction itself can

be object of intentional design is an important one, and my experiments indicate several essential characteristics of interaction in spatial computing.

Full vs. limited

The ways a system appears to be usable are often called its “affordances” [Gibson, 1979]. The affordances of a successful spatial computation system must be implemented so fully that there are no invisible barriers to its operation that disturb the illusion it is working to create.

My experiments achieved various levels of fullness. The affordances of *Installation* were the handling of the screen and the stylus. These were both heavily cabled, and therefore fell short of the fulfillment of free manipulation. *EyeBox*’s shortcomings were in the objects it did not handle well. The fullness of the interaction was disturbed and questions of the system’s competence were unavoidable. *LittleVision* was perhaps the fullest project by these standards. It was entirely self-contained, and it did everything a user might expect it to.

Feedback vs. one-way control

Feedback is essential to human control. The levels and kinds of feedback offered by spatial systems dramatically influence their usability.

Immediacy of feedback is the single most important quality of interaction. We are able to control our operations in the world only relative to feedback we receive about how they are proceeding. If a system limits such feedback it becomes difficult to control. This was apparent in almost every experiment.

Predictability vs. inconsistency

The second ingredient in accommodating human control is not frustrating expectation. A user’s desire to control a system should require as little conscious effort to achieve as possible.

This demands total consistency in operation and gratification of expected behavior. These are the foundations that allow us to plan action based on intention. This does not preclude surprise as an element, but too much surprise in the control of a system is just uncontrollability.

However, as demonstrated by *Stomping Ground*, perfect controllability is not always the goal. The system itself seemed to have something of a will of its own. Users knew they were effecting the outcome, but it was not quite clear how literally or directly. For a piece that had a strong artistic identity of its own, this turned out to be quite appropriate. It engaged users in a dialog with the system through which they were trying to understand their relationship to the sound and graphics.

14.3 Shortcuts

Spatial computing is starting to seem like a lot of work. But there are welcome shortcuts that help make such systems possible to produce without years of effort and outrageous expense.

Relativity of perception

The relativity of sensory experience is something to leverage. Since we have no absolute benchmarks it is often not necessary to calibrate to a hard external standard, which can be extremely difficult. This excludes certain applications that demand calibrated precision, such as surgical enhancement. But many fields (entertainment, design, visualization, for instance) remain open to a system that delivers the experience of real spatial interaction without being absolutely true.

Opacity

Some systems should become transparent—essentially unnoticeable to their users. Some should remain solid and visible. There is no hard rule, contrary to some opinions [Norman, 1998], that say all successful systems become transparent. Much depends on the intended focus of user attention. In many cases the system itself is part of what should be experienced. The extent to which a system should assert its presence must be considered and controlled closely by its designer.

Some systems exist to disappear. That was true of *Installation*, for instance. The screen was to be seen through. By contrast in *EyeBox*, the screen had an important physical identity as a door to a cabinet. It did not attempt to disappear. *LittleVision* always remained present. Its presence as an object and its unwillingness to render a photographic image kept it from disappearing. *Stomping Ground* was very much a solid presence. It was meant to be seen.

Many virtual environments and systems for augmented reality see transparency as their goal. This may be a design consideration for the specific project, but it is not intrinsic to the field. Visibility is part of a system's not apologizing for its existence. Since no system yet has become so seamlessly integrated into our sensory apparatus that it truly disappears, it behooves systems to deal responsibly with their visibility rather than pretend to be invisible and fail.

14.4 The Future of Spatial Computing

Technologies

We are beginning to see issues of space and interface enter the mainstream computer market as wireless networking and the tablet PC become popular. These technologies respectively encourage mobility and treatment of the screen as a literal space. It remains, however, a planar barrier.

Display

OLED and other display technologies are going to make digital displays larger, thinner, lighter, brighter, and more flexible. We are going to see them augment many fixed-appearance surfaces such as desks, walls, ceilings, and billboards. There is an opportunity for these surfaces to become more than fixed surfaces, to fold, open, wrap, flip, stretch, bend, hinge, or twist. I hope these potentials are realized.

Another class of display that is becoming available is the 3D display. The Spatial Imaging group at the Media Lab has a new 3D display that is capable of showing viewers full-color hologram-like images from digital files. The illusion of depth is very good. Displays like this one, particularly if they can be transparent where there is no object, have the potential to blend the real and virtual quite powerfully.

Tracking

Tracking objects in space must become more reliable and affordable before we see it used commonly. GPS works well outdoors, but it is useless inside or for precise location. Improvements will come from either advances in computer vision



Figure 14.1. Some laptops now allow their screens to be flipped and folded down to make the machine into a tablet controlled with a stylus. This opens up a whole variety of spatial interactions.



Figure 14.2. An image of a vice as seen on the new 3D display from the Spatial Imaging Group at the MIT Media Lab. (The vice was scanned with EyeBox.)

algorithms or new physical technologies for spatial sensing.

Scanning

Scanning objects in 3D will become faster, cheaper, and easier. It is unlikely that 3D scanners will be popular devices for the home unless there is a compelling application other than design. Most people do not own flatbed scanners today.

Printing

3D printing will become tremendously valuable as the process speeds up and the machines and materials drop in price. There is no reason that many objects that are primarily formal cannot be sold as digital representations and then printed locally.

Applications

As the technologies improve we will begin to ask more of our interactions with our machines. We will demand to be treated as more than ten fingers and a wrist.

Rather than try to augur the entire future of the discipline, let me paint a picture of a single application: a workplace with a spatially-sensitive network “ZapNet” in place.

Every employee is issued a laser-pointer-like device called a ZapStick. This is an enhanced pointable device capable of sending data over its beam and receiving it from a wireless LAN. Every display device in the office, be it a projection surface, laptop or desktop computer screen is outfitted with sensors to detect and read the ZapStick beam.

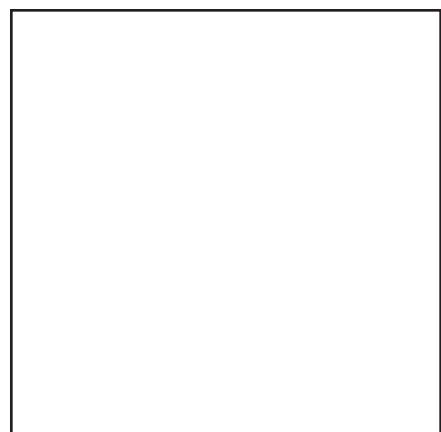


Figure 14.3. The ZapStick.

Any information being displayed can be sucked up into a ZapStick or squirted out onto a display by

one (provided the user is authorized to have it). Meeting notes can be shot out to participants as they arrive. Notes and files can be shot between machines during the meeting. Further, participants can scribble on the projection with their pointers during the meeting, and that writing will be saved and incorporated as an annotation layer on the file.

Large stores of information could be associated with physical objects around the office. In addition to a networked server with a specific name, an artifact such as a building model in an architectural office could be the locus of all the digital information pertaining to the project. The ZapSticks could be used to pull information from this site or send information to it. Groups of these models could be gathered together and brought to meetings or conversations where they are germane. Of course all of these digital interactions and sharing of information is possible without physical analogs, but it is what a technology makes easier that it promotes. In this case, the augmentation of a data network with a physical network promotes the flow of digital information on top of existing social interactions. We tie the space of data flow closer to the space of conversation.

This is one small idea for spatial computation. It is all too easy to imagine others. The real learning will come in building them.

15. References

- [Absolut, 2002] WindowsVR, www.abs-tech.com/Produtos/3D_VR/VR-Hardware/hmds1/Virtual_Research/win_vr.html, 2002.
- [Apple, 1996] Apple Computer's *Hotsauce*, http://www.inxight.com/news/apple_initiative.html.
- [Art+Com, 1997] Art+Com, *Virtual Car* for Daimler-Benz AG, 1997, www.artcom.de.
- [Arthur, 1993] Arthur, K. W., Booth, K. S., and Ware, C., *Evaluating 3D task performance for fish tank virtual worlds*, ACM Transactions on Information Systems, vol. 11, no. 3, July 1993, pp. 239-265.
- [Azuma, 1997] Azuma, R., T., *A survey of Augmented Reality*, in *Presence: Teleoperators and Virtual Environments*, 6, 4, 1997, p. 355-385.
- [Bachelard, 1964] Bachelard, G., *The Poetics of Space*, Jolas, M., trans., The Orion Press, 1964.
- [Barabási, 2002] Barabási, A., *Linked: The New Science of Networks*, Perseus Publishing, 2002.
- [Baudrillard, 1988] Baudrillard, J., *The Ecstasy of Communication*, New York: Semiotext(e), 1988.
- [Billinghurst, 2002] Billinghurst, M. and Kato, H., *Collaborative Augmented Reality*. Communications of the ACM, 45(7), 2002, pp. 64-70.
- [Boys, 1959] Boys, C.V., *Soap Bubbles, Their Colors and Forces which Mold Them*. Dover Publications, 1959.
- [Brooks, 2001] Brooks, R., *The Relationship Between Matter and Life*, in Nature 409, pp. 409-411; 2001.
- [Brown, 1997] Brown, K. N., *Grammatical design*, IEEE Expert: Intelligent Systems and their Applications 12, 1997, pp. 27-33.
- [Buxton, 1997] Buxton, W., *Living in Augmented Reality: Ubiquitous Media and Reactive Environments*. In Finn, K., Sellen, A., and Wilber, S. (Eds.). Video Mediated Communication. Hillsdale, N.J.: Erlbaum, 1996; pp. 363-384.
- [Card, 1996] Card, S. K., Robertson G. G., York W., *The WebBook and the Web Forager: An information workspace for the World Wide Web*. Proc CHI '96 ACM Conference on Human Factors in Computing Systems. New York: ACM Press, 1996; pp. 111-116.
- [Carroll, 2001] In *Human-Computer Interaction in the New Millennium*, John M. Carroll, ed.; Addison-Wesley, August 2001, pp. 579-601.
- [Colby, 1992] Colby, G. and Scholl, L., *Transparency and Blur as Selective Cue for Complex Information*. Proceedings of SPIE'92.

- [Conroy, 2002] Dalton, R. C., *Is Spatial Intelligibility Critical to the Design of Large-scale Virtual Environments?* International Journal of Design Computing, vol. 4, 2002, <http://www.arch.usyd.edu.au/kcdc/journal/vol4/dalton>.
- [Cubitt, 1998] Cubitt, S., *Digital Aesthetics*, SAGE Publications, Ltd., London, 1998.
- [Deering, 1992] Deering, M., *High resolution virtual reality*, in Computer Graphics, 26, 2, 1992, pp. 195-202.
- [Donath, 1995] Donath, J., *Visual Who: Animating the affinities and activities of an electronic community*, ACM Multimedia 95 - Electronic Proceedings, November 5-9, 1995, San Francisco, California.
- [Dorsey, 1996] Dorsey , J. and Hanrahan, P., *Modeling and Rendering of Metallic Patinas*. Proc. of SIGGRAPH '96. In *Computer Graphics Proceedings, Annual Conference Series*, 1996, ACM SIGGRAPH, pp. 387-396.
- [Dourish, 2000] Dourish, P., Edwards, W.K., et. al., *Extending Document Management Systems with User-Specific Active Properties*. ACM Transactions on Information Systems, 2000. 18(2), pp. 140-170.
- [Dunne, 2001] Dunne, A. and Raby, F., *Design Noir, the Secret Life of Electronic Objects*, August / Birkhäuser, Berlin, 2001.
- [Electronic Arts, 2003] Electronic Arts, *Sims Online*, 2003, <http://www.eagames.com/official/thesimsonline/home/index.jsp>.
- [Feiner, 1997] Feiner, S., MacIntyre, B., Höllerer, T., and Webster, A., *A Touring Machine: Prototyping 3D Mobile Augmented Reality Systems for Exploring the Urban Environment*. In Proc. ISWC '97 (Int. Symp. on Wearable Computers), pp. 74–81, 1997.
- [Fitzmaurice, 1993] Fitzmaurice, G. W., *Situated information spaces and spatially aware palmtop computers*, Communications of the ACM, Special issue on Augmented Reality and UbiComp, July 1993, 36(7), p.38-49.
- [Fry, 1999] Ben Fry, *Anemone*, 1999, <http://acg.media.mit.edu/people/fry/anemone>.
- [Gibson, 1979] Gibson, J.J., *The Ecological Approach to Visual Perception*. Houghton Mifflin,Boston, 1979.
- [Gombrich, 1969] Gombrich, E. H., *Art and Illusion, a study in the Psychology of Pictorial Representation*, Princeton University Press, Princeton, NJ, 1969.
- [Hayles, 1999] Hayles, N. K., *How We Became Posthuman*, The University of Chicago Press, Chicago & London, 1999.
- [IrDA, 2000] IrDA, *Infrared Data Association, Point and Shoot Profile*, March 20, 2000, <http://www.irda.org/standards/specifications.asp>.
- [Ishii, 1997] Ishii, H. and Ullmer, B., *Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms*, in Proc. of

- Conference on Human Factors in Computing Systems (CHI '97), (Atlanta, March 1997), ACM Press, pp. 234-241.
- [Karahalios, 1998] Karahalios, K., *Diorama*. in Conference abstracts and applications: SIGGRAPH '98, ACM Press, New York, p. 297.
- [Kline, 1999] C. Kline and B. Blumberg, *The Art and Science of Synthetic Character Design*. Convention of the Society for the Study of Artificial Intelligence and the Simulation of Behavior (AISB), Symposium on AI and Creativity in Entertainment and Visual Art, Proceedings, Edinburgh, Scotland, April, 1999.
- [Kuzu , 2002] Kuzu, Y., and Rodehorst, V., *Volumetric Modeling Using Shape From Silhouette*, 2002, www.fpk.tu-berlin.de/forschung/sonder/pub/DT4_kuzu.pdf.
- [Laibowitz, 2003] Laibowitz, M., Paradiso, J., *Phenomenological Model for Distributed Systems*, <http://www.media.mit.edu/resenv/phenom>, 2003.
- [Le Corbusier, 1923] Le Corbusier, *Towards a New Architecture*, 1923, Dover Pubns; (February 1986).
- [Linden Labs, 2003] Linden Labs, *Second Life*, 2003, <http://lindenlab.com>.
- [Loos, 1908] Loos, A., *Ornament and Crime*, 1908, in *Ornament and Crime: Selected Essays*, Mitchell, M. trans., Ariadne Press, 1998.
- [Manovich, 1996] Manovich, L., *The Aesthetics of Virtual Worlds: Report from Los Angeles*, CTHEORY, www.manovich.net, 1996.
- [Manovich, 2002] Manovich, L., *Generation Flash*, www.manovich.net, 2002.
- [Matsuoka, 2002] Matsuoka, H., Onozawa, A., Sato, H., Nojima, H., *Regeneration of Real Objects in the Real World*, Conference Abstracts and Applications of SIGGRAPH 2002, p.77,p.243.
- [Matusik, 2001] Matusik, W., Buehler, C., and McMillan, L., *Polyhedral Visual Hulls for Real-Time Rendering*, in Proc. Twelfth Eurographics Workshop on Rendering, 2001, pp. 115-125.
- [Matusik, 2000] Matusik, W., Buehler, C., Raskar, R., Gortler, S., and McMillan, L., *Image-based Visual Hulls*, in Proc. SIGGRAPH, 2000, pp. 369-374.
- [Milgram, 1999] Milgram, P., Colquhoun, H., *A Taxonomy of Real and Virtual World Display Integration*, in *Mixed Reality*, (Ohta, Y., Tamura, H., eds.), Ohmsa Ltd., Tokyo, 1999.
- [Nakayama, 1985] Nakayama, K., *Biological image motion processing: A review*. *Vision Research*, 25, 1985 , 625-660.
- [Norman, 1998] Norman, D., *The Invisible Computer*, MIT Press, Cambridge, MA, 1998.
- [Paquette, 2001] Paquette, E., Poulin, P., Drettakis, G., *Surface Aging by Impacts*, Proceedings of Graphics Interface 2001 , June 2001.

- [Paradiso, 1997] Paradiso, J., Abler, C., Hsiao, K. Y., Reynolds, M., *The Magic Carpet: Physical Sensing for Immersive Environments*, in Proc. of the CHI '97 Conference on Human Factors in Computing Systems, Extended Abstracts, ACM Press, NY, pp. 277-278 (1997).
- [Paradiso, 1999] Paradiso, J., *The Brain Opera Technology: New Instruments and Gestural Sensors for Musical Interaction and Performance*, Journal of New Music Research, 28(2), 1999, pp. 130-149.
- [Paradiso, 2000] Paradiso, J., Hsiao, K., Benbasat, A., Teegarden, Z., *Design and Implementation of Expressive Footwear*, IBM Systems Journal, Volume 39, Nos. 3 & 4, October 2000, pp. 511-529.
- [Pausch, 1997] Pausch, R., Proffitt, D., and Williams, G., *Quantifying immersion in virtual reality*, SIGGRAPH'97.
- [Pessoa, 2003] Pessoa, L., de Weed, P., eds., *Filling-in: From Perceptual Completion to Cortical Reorganization*, Oxford University Press, 2003.
- [Petitjean, 1998] Petitjean, S., *A Computational Geometric Approach to Visual Hulls*, Int. J. of Computer. Geometry and Appl., vol. 8, no.4, pp. 407-436, 1998.
- [Piekarski , 2002] Piekarski, W., and Thomas, B. H., *Unifying Augmented Reality and Virtual Reality User Interfaces*, Technical report, January 2002, University of South Australia.
- [Rekimoto, 1995] Rekimoto, J., *The Magnifying Glass Approach to Augmented Reality Systems*, International Conference on Artificial Reality and Tele-Existence '95 / Conference on Virtual Reality Software and Technology (ICAT/VRST '95).
- [Robertson, 1997] Robertson, G., Czerwinski, M., and van Dantzich, M., *Immersion in Desktop Virtual Reality*, UIST'97.
- [Saunders, 2001] Saunders, R., *Simplified ART*, <http://www.arch.usyd.edu.au/~rob/java/applets/neuro/SimplifiedARTDemo.html>, August 2001.
- [Selfridge, 1999] Selfridge, P. and Kirk, T., *Cospace: Combining Web-Browsing and Dynamically Generated 3D Multiuser Environments*, SIGART 10, 1, 1999, pp. 24-32.
- [Shiffman, 2000] Jared Shiffman, *Honey*, 2000, <http://acg.media.mit.edu/people/jarfish/honey>.
- [Slater, 2002] Slater, M., *Presence and the Sixth Sense*, in PRESENCE: Teleoperators and Virtual Environments, MIT Press, 11(4) , 2002, pp. 435-439.
- [Sullivan, 1918] Sullivan L., *The Tall Office Building Artistically Considered*, in: Athey I., ed. *Kindergarten Chats (revised 1918) and Other Writings*. New York 1947: 202-13.
- [Sullivan, 1998] Sullivan, S., and Ponce, J., *Automatic Model Construction, Pose Estimation, and Object Recognition from Photographs Using Triangular Splines*, IEEE Transactions on Pattern Analysis and

Machine Intelligence, 20(10):1091-1096, 1998.

[Szeliski, 1993]

Szeliski, R., *Rapid Octree Construction from Image Sequences*, CVGIP: Image Understanding, 58, 1 (July 1993), pp. 23-32.

[There, 2003]

There, Inc., *There*, 2003, <http://www.there.com>.

[Tsai, 1987]

Tsai, R. Y., *A versatile camera calibration technique for high accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses*, IEEE Trans., 1987, RA-3, pp. 323-344.

[Tsang, 2002]

Tsang, M., Fitzmaurice, G., Kurtenbach, G., Khan, A. and Buxton, W., Boom Chameleon: Simultaneous capture of 3D viewpoint, voice and gesture annotations on a spatially-aware display. Alias|Wavefront. Submitted for publication, 2002. [<http://www.billbuxton.com/boomChameleon.pdf>]

[Ullmer, 1997]

Ullmer B., Ishii H., *The metaDESK: Models and Prototypes for Tangible User Interfaces*. Proc. of UIST'97, pp.223-232.

[Ullmer, 1998]

Ullmer, B., et al., *mediaBlocks: Physical Containers, Transports, and Controls for Online Media*, in Proceedings of SIGGRAPH '98, ACM Press, 1998, pp. 379-386.

[Ullmer, 2000]

Ishii, H. and Ullmer, B., *Emerging Frameworks for Tangible User Interfaces*, in IBM Systems Journal Volume 39 Nos. 3&4, 2000; pp. 915-931.

[Van den Berg, 1955]

Van den Berg, J. H., *The Phenomenological Approach in Psychology*, Charles C. Thomas, publisher. Springfield, Illinois, 1955, p. 61. Quoted in [Bachelard, 1964; xvii.]

[Ventrella, 1997]

Ventrella, J., *Arrows in the Fluid Mind, A 2D Fluid Dynamics Model for Animation Based on Intuitive Physics*, 1997 , <http://www.ventrella.com/Ideas/Fluid/fluid.html>.

- [Venturi, 2001] Venturi, R., *Robert Venturi's Disorderly Ode*, Metropolis, Sept. 2001.
- [Weiner, 1948] Weiner, N., *Cybernetics: Or Control & Communication in the Animal and the Machine*, MIT Press, Cambridge, MA, 1948.
- [Weiser, 1988] <http://www.ubiq.com/hypertext/weiser/UbiHome.html>
- [Witmer, 1998] Witmer, B.G., Singer, M.J., *Measuring Presence in Virtual Environments: A Presence Questionnaire*, *Presence*, 7 (3), 1998, pp. 225-240.
- [Yarbus, 1967] Yarbus, A. L., *Eye Movements and Vision*. Plenum Press, New York, 1967.
- [Zhai, 1996] Zhai, S., Buxton, W., and Milgram, P., *The partial-occlusion effect: Utilizing semitransparency in 3D human-computer interaction*, in *ACM Transactions on Computer-Human Interaction*, 3(3), 254-284.

Appendix A: Other Experiments

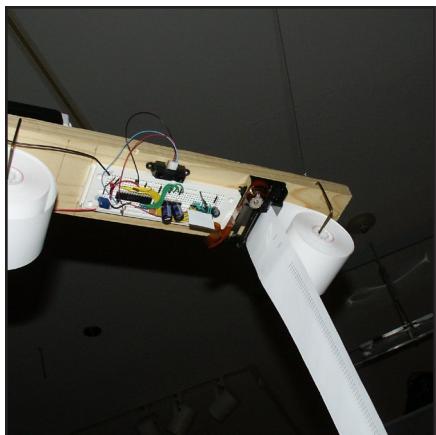


Figure A.1. Two photos of *Tear Me* as installed in a gallery. The proximity sensor is at the top edge of the board here.

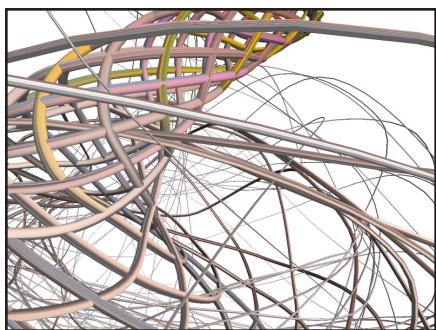


Figure A.2. Weaver explored the forms of woven and braided strands.

The six projects featured in the body of the thesis are those that best describe and comment on spatial computing. In the time of my thesis work, however, I completed a number of other projects with close ties to my core ideas. I describe some of them here.

Tear Me

Tear Me is a line of six thermal print heads suspended from the ceiling. When people walk below it, they are sensed with infra-red proximity sensors, and the printers near them print down at them. Each printer repeats one word over and over, and together they form a sentence: “let” “me” “know” “if I’m” “reaching” “you.” Viewers are encouraged to tug on the paper, and to tear off excess as it gets long and drop it on the floor at their feet.

Weaver

Weaver is a dynamic formal exploration of woven and braided patterns as a generator of architectural form. It grew out of a project I began with Professor Peter Testa when I was in the Architecture department prior to my work at the Media Lab. He was exploring woven carbon fiber as a structural material for building, and I did some work creating grammars for encoding weaves and writing software to render the patterns.

Two years later, while at the Media Lab, I had the opportunity to work with Testa again to update my work, making it a dynamic, real-time, somewhat interactive system. Users rolled a trackball to orient the growing shape in space or click one of its buttons to freeze the form for a few seconds for closer inspection. The form itself was an evolving weave that shots around the screen, intersecting with itself, exploding toward the user, and creating patterns of seemingly tremendous complexity out of

a few simple geometrical rules. The shapes depend both on the description of the weave pattern and the topology of the surface on which the weave is applied.

Active Statics

I worked with professors emeritus Ed Allen and Waclaw Zalewski to create a set of eight highly interactive demonstrations for teaching elementary structural design to architecture or engineering students. For maximum transparency and creative potential, these demonstrations were based on graphic statics, a body of techniques used by such masters as Antonio Gaudi, Gustave Eiffel, and Robert Maillart to create their structural masterpieces.

The external forces on each structure are plotted to a scale of length to force on a load line. Working from the load line, the forces in the members of the structure are determined by scaling the lengths of lines constructed parallel to the members. The diagram of forces that results from this process is called the force polygon. In these demonstrations, the force polygon is constructed automatically and revised instantly and continually as changes are made in the form or loading of the structure. The intensities of member forces are indicated by the lengths of the line segments in the force polygon, the numerical values in a table, and by the thicknesses of the members themselves. All components of each demonstration are color-coded: Blue indicates tension, red is compression, yellow is zero force. External loads are black and reactions are green.

What makes these demos important is their feedback. Students are able to see analytical feedback as a direct consequence of formal manipulation. The directness of this feedback loop is unprecedented, and it is a tremendous aid to the development of intuition. Previously, the analysis stage had to happen subsequent to design, and the two processes were separate. There was a practical

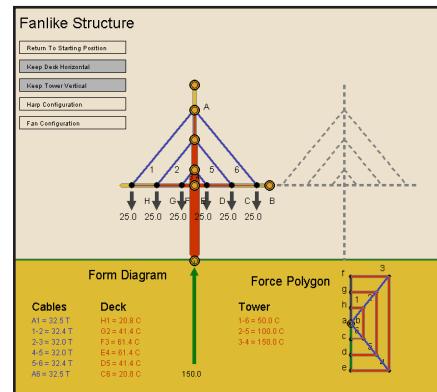
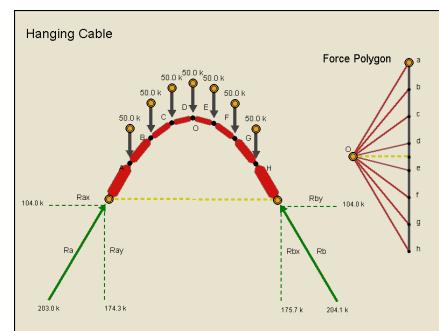
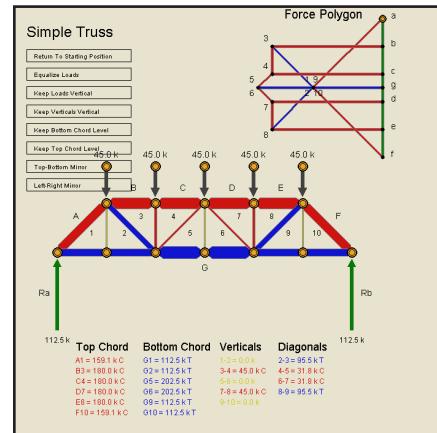


Figure A.3. Three of the eight interactive teaching demos. Any yellow point can be manipulated. The analysis updates in real time.

limit to the number of times any designer would modify a design and re-run an analysis. Only with the analysis baked directly into the process of design can the number of iterations become essentially infinite.

The demos are currently in use for courses at several universities. They are publicly available at acg.media.mit.edu/people/simong/statics/data/.

Moment

Moment is a pure parametric design two-dimensional design environment. It implements a visual data-flow language for the prototyping of dynamic graphics. The process of designing in *Moment* is one of connecting terminals with wires, spinning them together into bundles when necessary.

Moment is the last incarnation of a project that has had many aliases and iterations, first as a series of two constraint-based languages called *RunningThings* and *Paragon*, next as a visual data-flow language, *Paramour*, and now finally, revised, retooled, and expanded as *Moment*.

Moment is available for download at acg.media.mit.edu/people/simong/moment/.

High Definition

High Definition is a piece that like *Internaut* maps an undimensioned space to a virtual space. In this case the space of origin is the dictionary. The piece begins with a single dictionary definition of word chosen in advance. The definition trails the word, flapping and waving in a circle in 3D space on the screen. Every so often one of the words of the definition quivers and explodes into its own definition, starting a new circle. This process continues recursively, creating a boiling tangle of words and definitions. The user can explore this definition space at will.

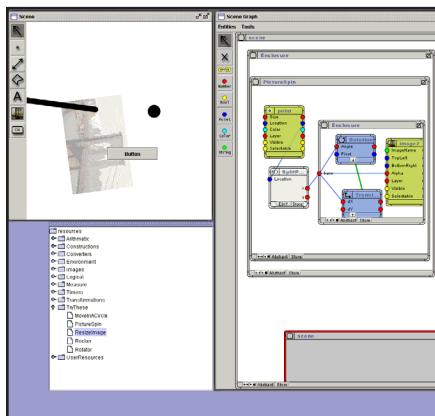


Figure A.4. The visual data-flow language *Moment*.

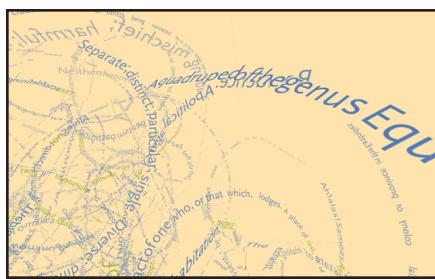
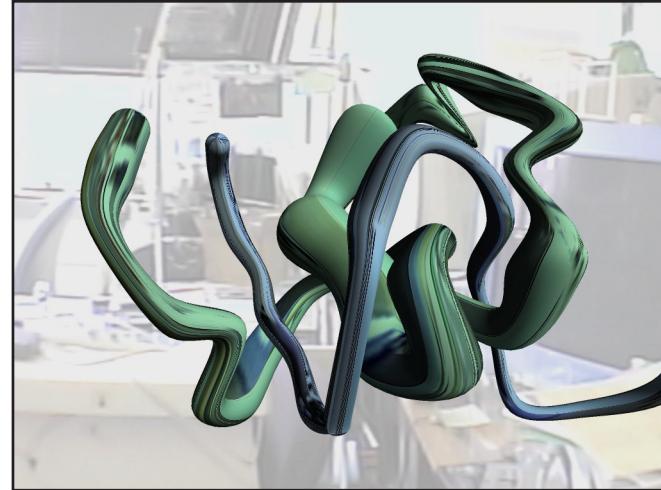


Figure A.5. *High Definition* explored the space of dictionary definitions.

Appendix B: SBalls Code

SBalls are useful for making organic-looking blobby tube shapes like the ones on the right. They are implemented in C++ and require OpenGL and GLUT.



SBalls.h

```
#ifndef _SBALLS_H_
#define _SBALLS_H_

#define BALL_SLICES 30
#define FATNESS_MULTIPLIER 1
#define BALL_STACKS 10
#define CAP_STACKS 20

#define TOTAL_STACKS (((nNodes-1)*BALL_STACKS) + 2 * CAP_STACKS +1)
#define VERTEX(slice, stack) (fullVertices+((slice)*TOTAL_STACKS+(stack))*3))
#define VERTEX_INDEX_GL(slice, stack) ((slice)*TOTAL_STACKS+(stack))
#define NORMAL(slice, stack) (normals+((slice)*TOTAL_STACKS+(stack))*3))
#define QUAD_SLICE(slice) (quads+((slice)*TOTAL_STACKS*2))
#define NODE_VERTEX(slice, node) (nodeVertices+((slice)*nNodes+(node))*3))

enum {BALLS_AND_WIRES = 0, SURFACES, BALLS_AND_SURFACES, WIRES};

// A node is a sphere in the
struct SBallNode {
    // Settable
    float    pos[3];      // Starting node location
    float    dPos[3];     // Move this much on each update
    float    radius;       // Starting radius
    float    dRadius;      // Amount of radius change per update

    // Internal
    SBallNode *next;
    SBallNode *previous;
    int        rotOffset; // Used to control twisting of outside surface
};

class SBalls {
public:

    // Public properties
    int displayMode;          // How to draw: BALLS_AND_WIRES, SURFACES, BALLS_AND_SURFACES, or WIRES
    float    color[4];         // Base color, RBGA
    float    offset[3];        // Offset of blob
    bool     useTexture;       // Texture-map onto blob or not

    // Construct/desctruct
    SBalls();
    ~SBalls();

    // Add nodes
    void appendNode(float pos[3], float radius);
    void appendNode(float posX, float posY, float posZ, float radius);
    void appendNode(float posX, float posY, float posZ, float radius, float dX, float dY, float dZ, float dR);

    void getExtents(float *xMin, float *xMax, float *yMin, float *yMax, float *zMin, float *zMax);

    void draw();               // Issue the GL calls to draw the SBalls object
                               // If you're using a texture, you need to set it up before calling this
                               // with glTexImage2D or some such call
};
```

```

// Updates
void update();           // Update node positions and sizes and recalculate every surface point (slow)
void silentUpdate();     // Just update node positions and sizes (fast)
void calculate();         // Recalculate every surface point (slow)

int nNodes;

// The ends of the nodelist
SBallNode *startNode;
SBallNode *endNode;

private:

float      *nodeVertices;          // Data structure as straight vertices: SLICES * NODES * 3
unsigned int *quads;              // Data structure as quad strips SLICES * ((NODES * STACKS) + 1) * 2
float      *fullVertices;         // Data structure as straight vertices: SLICES * ((STACKS * NODES) + 1) * 3
float      *normals;              // Data structure SLICES * NODES * STACKS * 3;
float      *texCoords;            // Data structure SLICES * NODES * STACKS * 2;

SBallNode* previousIfExists(SBallNode *cur);
SBallNode* nextIfExists(SBallNode *cur);

// Utility functions
float distSq(float vec1[3], float vec2[3]);
void vecCross(float vec1[3], float vec2[3], float result[3]);
void vecCopy(float in[3], float out[3]);
void vecMinus(float a[3], float b[3]);
void vecPlus(float a[3], float b[3]);
void vecTimes(float a[3], float b);
void vecAverage(float a[3], float b[3]);
void vecAverage(float fromA[3], float toA[3], float fromB[3], float toB[3], float average[3]);
float vecMag(float vec[3]);
void vecNormalize(float vec[3]);
void vecRotate(float vec[3], double theta, float axis[3]);
void catmullromPoint(int upToNode, int slice, float t, float curvePoint[3]);
float catmullromBasis(int i, float t);
void fillQuadStructure();
void fillVertexData();
void makeNormals();

float tempVec[3];
float averageVec[3];
float averageVecOrigin[3];
float tempVec2[3];
float tempVec3[3];
float circleAxis[3];
float tempVec4[3];
float tempVec5[3];
float inFromPrevious[3];
float inFromNext[3];
};

#endif

```

SBalls.cpp

```
/*-----*/
/* SBalls forms a texture-mapped blobby surface as a series of
/* spheres of different radius located along a curve and connected by
/* Catmull-Rom splines.
*/
/*
/* Documentation in header.
*/
/*
/* 10/07/01 Simon Greenwold
/* Modified 5/17/03
/*-----*/

#include "SBalls.h"
#include <GL/gl.h>
#include <GL/glut.h>
#include <float.h>
#include <math.h>

/* Some <math.h> files do not define M_PI... */
#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif

#ifndef MIN
#define MIN(x,y) (((x) < (y)) ? (x) : (y))
#endif

#ifndef MAX
#define MAX(x,y) (((x) > (y)) ? (x) : (y))
#endif

SBalls::SBalls() {
    startNode = NULL;
    endNode = NULL;
    nNodes = 0;
    displayMode = BALLS_AND_WIRES;
    color[0] = 0.0;
    color[1] = 1.0;
    color[2] = 0.0;
    color[3] = 1.0;
    offset[0] = 0.0;
    offset[1] = 0.0;
    offset[2] = 0.0;
    useTexture = FALSE;
}

SBalls::~SBalls() {
    SBallNode *cur = startNode;
    SBallNode *next;
    while (cur != NULL) {
        next = cur->next;
        delete(cur);
        cur = next;
    }
    if (nNodes > 1) {
        delete nodeVertices;
        delete quads;
        delete normals;
        delete fullVertices;
        delete texCoords;
    }
}

void SBalls::getExtents(float *xMin, float *xMax, float *yMin, float *yMax, float *zMin, float *zMax) {
    *xMin = FLT_MAX;
    *xMax = FLT_MIN;
    *yMin = FLT_MAX;
    *yMax = FLT_MIN;
    *zMin = FLT_MAX;
    *zMax = FLT_MIN;

    float x, y, z, r;

    for (SBallNode* nodePtr = startNode; nodePtr != NULL; nodePtr = nodePtr->next) {
        r = nodePtr->radius;

        x = nodePtr->pos[0];
        if (x - r < *xMin)
            *xMin = x - r;
        if (x + r > *xMax)
            *xMax = x + r;

        y = nodePtr->pos[1];
        if (y - r < *yMin)
            *yMin = y - r;
        if (y + r > *yMax)
            *yMax = y + r;
    }
}
```

```

    *yMax = y + r;

    z = nodePtr->pos[2];
    if (z - r < *zMin)
        *zMin = z - r;
    if (z + r > *zMax)
        *zMax = z + r;
}

void SBalls::appendNode(float pos[3], float radius) {
    appendNode(pos[0], pos[1], pos[2], radius);
}

void SBalls::appendNode(float posX, float posY, float posZ, float radius) {
    appendNode(posX, posY, posZ, radius, 0, 0, 0, 0);
}

void SBalls::appendNode(float posX, float posY, float posZ, float radius, float dX, float dY, float dZ, float dR) {
    nNodes++;
    SBallNode *newNode = new SBallNode;
    newNode->pos[0] = posX;
    newNode->pos[1] = posY;
    newNode->pos[2] = posZ;
    newNode->radius = radius;
    newNode->rotOffset = 0;

    newNode->dPos[0] = dX;
    newNode->dPos[1] = dY;
    newNode->dPos[2] = dZ;
    newNode->dRadius = dR;

    if (!startNode)
        startNode = newNode;
    newNode->next = NULL;
    newNode->previous = endNode;
    if (endNode)
        endNode->next = newNode;
    endNode = newNode;

    if (nNodes > 1) {
        delete nodeVertices;
        delete quads;
        delete normals;
        delete fullVertices;
        delete texCoords;
    }
    nodeVertices = new float[BALL_SLICES * nNodes * 3];
    quads = new GLuint [BALL_SLICES * TOTAL_STACKS * 2];
    fullVertices = new float [BALL_SLICES * TOTAL_STACKS * 3];
    normals = new float [BALL_SLICES * TOTAL_STACKS * 3];
    texCoords = new float [BALL_SLICES * TOTAL_STACKS * 2];

    fillQuadStructure();
    calculate();
}
}

void SBalls::calculate() {
    // Draw average vectors

    SBallNode *cur = startNode;
    if (!cur->next)
        return;

    float rotAngle = 360.0 / BALL_SLICES;
    int numNode = 0;
    int rotOffset = 0;
    while (cur) {
        if (cur == startNode) { // First node
            vecCopy(cur->pos, circleAxis);
            vecMinus(circleAxis, cur->next->pos);

            vecCopy(circleAxis, tempVec2);
            tempVec2[1] += 30;
            tempVec2[2] += circleAxis[2] + 40;

            vecCross(circleAxis, tempVec2, averageVecOrigin);
            vecTimes(averageVecOrigin, FATNESS_MULTIPLIER * cur->radius / vecMag(averageVecOrigin));
        }
        else if (cur == endNode) { // last node
            vecCopy(cur->previous->pos, circleAxis);
            vecMinus(circleAxis, cur->pos);

            vecCopy(circleAxis, tempVec2);
            tempVec2[1] += 30;
            tempVec2[2] += circleAxis[2] + 40;

            vecCross(circleAxis, tempVec2, averageVecOrigin);
        }
    }
}

```

```

        vecTimes(averageVecOrigin, FATNESS_MULTIPLIER * cur->radius / vecMag(averageVecOrigin));
    }
    else { // a node in the middle
        vecCopy(cur->pos, inFromPrevious);
        vecCopy(cur->next->pos, inFromNext);
        vecMinus(inFromPrevious, cur->previous->pos);
        vecMinus(inFromNext, cur->pos);

        vecAverage(cur->previous->pos, cur->pos, cur->next->pos, cur->pos, averageVecOrigin);

        vecCross(inFromPrevious, inFromNext, tempVec2);
        vecCross(averageVecOrigin, tempVec2, circleAxis);
        vecTimes(averageVecOrigin, FATNESS_MULTIPLIER * cur->radius / vecMag(averageVecOrigin));
    }

    // Get positions
    float tempPts[BALL_SLICES][3];

    rotOffset += cur->rotOffset;
    for (int slice = 0; slice < BALL_SLICES; slice++) {
        vecRotate(averageVecOrigin, rotAngle, circleAxis);
        vecCopy(averageVecOrigin, tempVec5);
        vecPlus(tempVec5, cur->pos);
        vecCopy(tempVec5, tempPts[slice]);
    }

    // Determine rotational offsets to prevent twisting
    float minCost = FLT_MAX;
    SBallNode* prev;
    float cost;

    if (cur != startNode) { // Offset of start is always zero
        prev = cur->previous;
        for (int rot = 0; rot < BALL_SLICES; rot++) {
            cost = 0.0f;
            for (int i = 0; i < BALL_SLICES; i++) {
                cost += distSq(tempPts[(i + rot) % BALL_SLICES], NODE_VERTEX(i, numNode - 1));
            }
            if (cost < minCost) {
                minCost = cost;
                cur->rotOffset = rot;
            }
        }
        // printf("Min cost %d: %f, %d\n", numNode, minCost, cur->rotOffset);
    }
    for (int i = 0; i < BALL_SLICES; i++) {
        vecCopy(tempPts[(i + cur->rotOffset) % BALL_SLICES], NODE_VERTEX(i, numNode));
    }

    cur = cur->next;
    numNode++;
}
if (nNodes > 1) {
    fillVertexData();
    makeNormals();
}
}

void SBalls::fillQuadStructure() {
    int nextSlice;
    GLuint* index;

    for (int slice = 0; slice < BALL_SLICES; slice++) {
        nextSlice = (slice + 1) % BALL_SLICES;
        index = QUAD_SLICE(slice);

        for (int stack = 0; stack < TOTAL_STACKS; stack++) {
            index[0] = VERTEX_INDEX_GL(slice, stack);
            index[1] = VERTEX_INDEX_GL(nextSlice, stack);

            texCoords[VERTEX_INDEX_GL(slice, stack) * 2] = ((float)stack) / (TOTAL_STACKS - 1) * (640.0f / 1024.0f);
            texCoords[VERTEX_INDEX_GL(slice, stack) * 2 + 1] = ((float)slice) / (BALL_SLICES - 1) * (480.0f / 512.0f);

            index += 2;
        }
    }
}

void SBalls::fillVertexData() {
    float t = 0.0f;
    float* index = fullVertices;
    float curvePoint[3];
    float stackInc = 1.0 / BALL_STACKS;
    for (int slice = 0; slice < BALL_SLICES; slice++) {

        if (nNodes > 1) {
            for (int capStack = 0; capStack < CAP_STACKS; capStack++) {

```

```

tempVec4[0] = NODE_VERTEX(slice, 0)[0] - startNode->pos[0];
tempVec4[1] = NODE_VERTEX(slice, 0)[1] - startNode->pos[1];
tempVec4[2] = NODE_VERTEX(slice, 0)[2] - startNode->pos[2];

tempVec5[0] = startNode->next->pos[0] - startNode->pos[0];
tempVec5[1] = startNode->next->pos[1] - startNode->pos[1];
tempVec5[2] = startNode->next->pos[2] - startNode->pos[2];

vecCross(tempVec5, tempVec4, tempVec3);
vecRotate(tempVec4, (90.0 / CAP_STACKS) * (CAP_STACKS - capStack), tempVec3);
VERTEX(slice, capStack)[0] = startNode->pos[0] + tempVec4[0];
VERTEX(slice, capStack)[1] = startNode->pos[1] + tempVec4[1];
VERTEX(slice, capStack)[2] = startNode->pos[2] + tempVec4[2];
}
}

for (int node = 1; node < nNodes; node++) {

    t = 0.0f;
    for (int stack = 0; stack < BALL_STACKS; stack++) {
        catmullromPoint(node, slice, t, curvePoint);
        VERTEX(slice, ((node - 1) * BALL_STACKS) + stack + CAP_STACKS)[0] = curvePoint[0];
        VERTEX(slice, ((node - 1) * BALL_STACKS) + stack + CAP_STACKS)[1] = curvePoint[1];
        VERTEX(slice, ((node - 1) * BALL_STACKS) + stack + CAP_STACKS)[2] = curvePoint[2];

        //printf("Catmull-rom: %f, %f, %f\n", index[0], index[1], index[2]);
        t += stackInc;
        index += 3;
    }
}

VERTEX(slice, (TOTAL_STACKS - 1))[0] = NODE_VERTEX(slice, nNodes - 1)[0];
VERTEX(slice, (TOTAL_STACKS - 1))[1] = NODE_VERTEX(slice, nNodes - 1)[1];
VERTEX(slice, (TOTAL_STACKS - 1))[2] = NODE_VERTEX(slice, nNodes - 1)[2];

if (nNodes > 1) {
    int i = 0;
    for (int capStack = (TOTAL_STACKS - 1) - CAP_STACKS; capStack < TOTAL_STACKS; capStack++) {
        tempVec4[0] = NODE_VERTEX(slice, (nNodes - 1))[0] - endNode->pos[0];
        tempVec4[1] = NODE_VERTEX(slice, (nNodes - 1))[1] - endNode->pos[1];
        tempVec4[2] = NODE_VERTEX(slice, (nNodes - 1))[2] - endNode->pos[2];

        tempVec5[0] = endNode->previous->pos[0] - endNode->pos[0];
        tempVec5[1] = endNode->previous->pos[1] - endNode->pos[1];
        tempVec5[2] = endNode->previous->pos[2] - endNode->pos[2];

        vecCross(tempVec4, tempVec5, tempVec3);
        vecRotate(tempVec4, (-90.0 / CAP_STACKS) * i, tempVec3);
        VERTEX(slice, capStack)[0] = endNode->pos[0] + tempVec4[0];
        VERTEX(slice, capStack)[1] = endNode->pos[1] + tempVec4[1];
        VERTEX(slice, capStack)[2] = endNode->pos[2] + tempVec4[2];
        i++;
    }
}
}

void SBalls::makeNormals() {
    int nextSlice;
    float around[3];
    float down[3];

    for (int slice = 0; slice < BALL_SLICES; slice++) {
        nextSlice = (slice + 1) % BALL_SLICES;
        for (int stack = 0; stack < TOTAL_STACKS - 1; stack++) {
            vecCopy(VERTEX(nextSlice, stack), around);
            vecMinus(around, VERTEX(slice, stack));

            vecCopy(VERTEX(slice, stack + 1), down);
            vecMinus(down, VERTEX(slice, stack));

            vecCross(down, around, NORMAL(slice, stack));
            //vecNormalize(NORMAL(slice, stack));
        }
        vecCopy(NORMAL(slice, TOTAL_STACKS - 2), NORMAL(slice, TOTAL_STACKS - 1));
    }
}

void SBalls::silentUpdate() {
    for (SBallNode* node = startNode; node; node = node->next) {
        vecPlus(node->pos, node->dPos);
        node->radius += node->dRadius;
        if (node->radius <= 0.0)
            node->radius = .05;
    }
}

void SBalls::update() {
    silentUpdate();
    calculate();
}

```

```

}

void SBalls::draw() {
    SBallNode *cur = startNode;

    glPushMatrix();
    glTranslatef(offset[0], offset[1], offset[2]);
    glLineWidth(1.2);

    glDisable(GL_TEXTURE_2D);

    // Draw balls
    if (displayMode == BALLS_AND_WIRES || displayMode == BALLS_AND_SURFACES) {
        glColor4f(1.0, 0.0, 0.0, 0.5);
        cur = startNode;
        while (cur) {
            glPushMatrix();

            glTranslatef(cur->pos[0], cur->pos[1], cur->pos[2]);
            glutWireSphere(cur->radius, 20, 20);

            cur = cur->next;
            glPopMatrix();
        }
    }

    if (nNodes <= 1) {
        glPopMatrix();
        return;
    }

    // Draw wires
    if (displayMode == BALLS_AND_WIRES || displayMode == WIRES) {

        //draw the fast way
        glEnableClientState(GL_VERTEX_ARRAY);
        glVertexPointer(3, GL_FLOAT, 0, fullVertices);
        glDisableClientState(GL_NORMAL_ARRAY);
        glDisableClientState(GL_TEXTURE_COORD_ARRAY);
        glDisable(GL_TEXTURE_2D);

        glColor4f(color[0], color[1], color[2], 0.5); //color[3]);
        for (int slice = 0; slice < BALL_SLICES; slice++) {
            glDrawArrays(GL_LINE_STRIP, VERTEX_INDEX_GL(slice, 0), TOTAL_STACKS);
        }
    }

    // Draw surfaces
    if (displayMode == BALLS_AND_SURFACES || displayMode == SURFACES) {

        glEnableClientState(GL_VERTEX_ARRAY);
        glVertexPointer(3, GL_FLOAT, 0, fullVertices);
        glEnableClientState(GL_NORMAL_ARRAY);
        glNormalPointer(GL_FLOAT, 0, normals);

        if (useTexture) {
            glEnableClientState(GL_TEXTURE_COORD_ARRAY);
            glTexCoordPointer(2, GL_FLOAT, 0, texCoords);
            glEnable(GL_TEXTURE_2D);
            glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
            glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
            glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
            glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
            glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
        }

        glColor4f(color[0], color[1], color[2], color[3]);

        for (int slice = 0; slice < BALL_SLICES; slice++) {
            glDrawElements(GL_QUAD_STRIP, TOTAL_STACKS * 2, GL_UNSIGNED_INT, QUAD_SLICE(slice));
        }

        glDisable(GL_TEXTURE_2D);
    }
    glPopMatrix();
}

//evaluate a point on the B spline
// i may vary from 2 to POINTS - 2
// beyond that interpolation breaks down
void SBalls::catmullromPoint(int upToNode, int slice, float t, float curvePoint[3]) {

/*float basis[4];
for (int j = -2; j<=1; j++){
    basis[j + 2] = catmullromBasis(j,t);
}

```

```

for (int i = 0; i < 3; i++) {
    curvePoint[i] = basis[0] * previousIfExists(previousIfExists(upToNode))>circlePoints[slice][i] +
        basis[1] * previousIfExists(upToNode))>circlePoints[slice][i] +
        basis[2] * upToNode->circlePoints[slice][i] +
        basis[3] * nextIfExists(upToNode))>circlePoints[slice][i];
}

float basis;
int index;
curvePoint[0] = 0.0f;
curvePoint[1] = 0.0f;
curvePoint[2] = 0.0f;

for (int j = -2; j<=1; j++){
    basis = catmullromBasis(j,t);
    index = MIN(MAX(upToNode + j, 0), nNodes - 1);
    float* point = NODE_VERTEX(slice, index);

//    if (slice == 2)
//        printf("Retrieving, slice, index, [0]: %i, %i, %f\n", slice, index, NODE_VERTEX(slice,index)[0]);
//        printf("In Catmull-rom: %f, %i, %f\n", basis, index, point[0]);
//        printf(" Still in Catmull-rom: %i, %i, %i\n", slice, nNodes, index);

    curvePoint[0] += basis * point[0];
    curvePoint[1] += basis * point[1];
    curvePoint[2] += basis * point[2];
}
}

// Catmull-Rom spline is just like a B spline, only with a different basis
float SBalls::catmullromBasis(int i, float t) {
    switch (i) {
        case -2:
            return ((-t+2)*t-1)*t/2;
        case -1:
            return (((3*t-5)*t)*t+2)/2;
        case 0:
            return ((-3*t+4)*t+1)*t/2;
        case 1:
            return ((t-1)*t*t)/2;
    }
    return 0; //we only get here if an invalid i is specified
}

SBallNode* SBalls::previousIfExists(SBallNode *cur) {
    if (cur->previous)
        return cur->previous;
    return cur;
}

SBallNode* SBalls::nextIfExists(SBallNode *cur) {
    if (cur->next)
        return cur->next;
    return cur;
}

float SBalls::distSq(float vec1[3], float vec2[3]) {
    return (vec1[0] - vec2[0]) * (vec1[0] - vec2[0]) +
        (vec1[1] - vec2[1]) * (vec1[1] - vec2[1]) +
        (vec1[2] - vec2[2]) * (vec1[2] - vec2[2]);
}

void SBalls::vecCross(float vec1[3], float vec2[3], float result[3]) {
    result[0] = (vec1[1] * vec2[2]) - (vec2[1] * vec1[2]);
    result[1] = (vec1[2] * vec2[0]) - (vec2[2] * vec1[0]);
    result[2] = (vec1[0] * vec2[1]) - (vec2[0] * vec1[1]);
}

void SBalls::vecCopy(float in[3], float out[3]) {
    out[0] = in[0];
    out[1] = in[1];
    out[2] = in[2];
}

void SBalls::vecMinus(float a[3], float b[3]) {      // a - b
    a[0] = a[0] - b[0];
    a[1] = a[1] - b[1];
    a[2] = a[2] - b[2];
}

void SBalls::vecPlus(float a[3], float b[3]) {      // a + b
    a[0] = a[0] + b[0];
    a[1] = a[1] + b[1];
    a[2] = a[2] + b[2];
}

void SBalls::vecTimes(float a[3], float b) {      // a * b
}

```

```

a[0] = a[0] * b;
a[1] = a[1] * b;
a[2] = a[2] * b;
}

void SBalls::vecAverage(float a[3], float b[3]) {
    vecPlus(a, b);
    vecTimes(a, 0.5);
}

void SBalls::vecAverage(float fromA[3], float toA[3], float fromB[3], float toB[3], float average[3]) {
    vecCopy(toA, average);
    vecMinus(average, fromA);
    vecCopy(toB, tempVec);
    vecMinus(tempVec, fromB);
    vecAverage(average, tempVec);
}

float SBalls::vecMag(float vec[3]) {
    return sqrt(vec[0] * vec[0] + vec[1] * vec[1] + vec[2] * vec[2]);
}

void SBalls::vecNormalize(float vec[3]) {
    vecTimes(vec, 1.0 / vecMag(vec));
}

/*
    Rotate a point p by angle theta around an arbitrary axis r
    Return the rotated point.
    Positive angles are anticlockwise looking down the axis
    towards the origin.
*/
void SBalls::vecRotate(float vec[3], double theta, float axis[3])
{
    float q[3] = {0.0,0.0,0.0};
    double costheta, sintheta;

    vecNormalize(axis);
    costheta = cos(theta * M_PI / 180.0);
    sintheta = sin(theta * M_PI / 180.0);

    q[0] += (costheta + (1 - costheta) * axis[0] * axis[0]) * vec[0];
    q[0] += ((1 - costheta) * axis[0] * axis[1] - axis[2] * sintheta) * vec[1];
    q[0] += ((1 - costheta) * axis[0] * axis[2] + axis[1] * sintheta) * vec[2];

    q[1] += ((1 - costheta) * axis[0] * axis[1] + axis[2] * sintheta) * vec[0];
    q[1] += (costheta + (1 - costheta) * axis[1] * axis[1]) * vec[1];
    q[1] += ((1 - costheta) * axis[1] * axis[2] - axis[0] * sintheta) * vec[2];

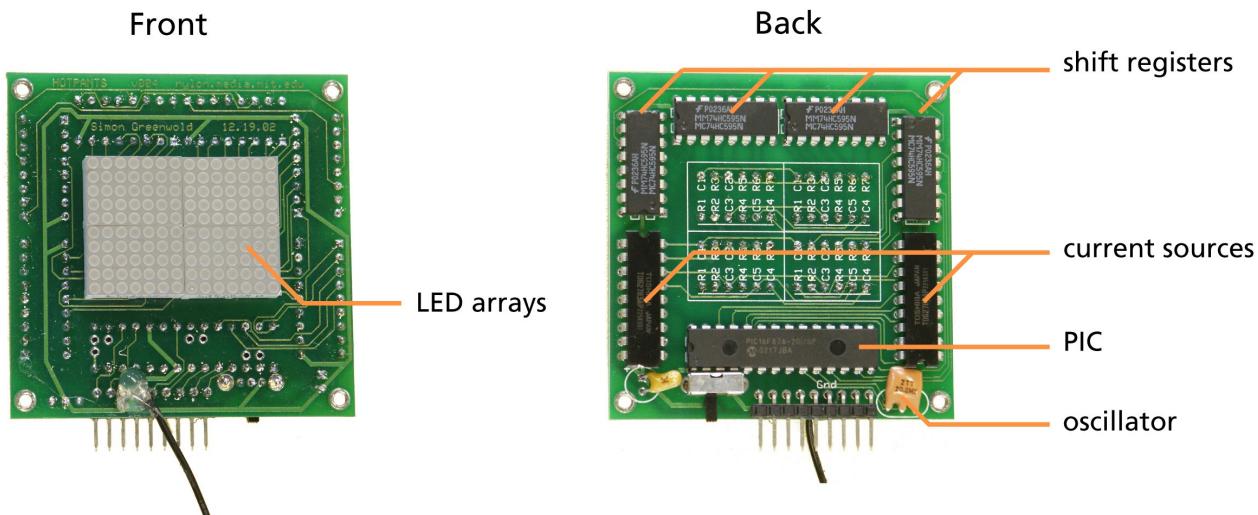
    q[2] += ((1 - costheta) * axis[0] * axis[2] - axis[1] * sintheta) * vec[0];
    q[2] += ((1 - costheta) * axis[1] * axis[2] + axis[0] * sintheta) * vec[1];
    q[2] += (costheta + (1 - costheta) * axis[2] * axis[2]) * vec[2];

    vec[0] = q[0];
    vec[1] = q[1];
    vec[2] = q[2];
}

```

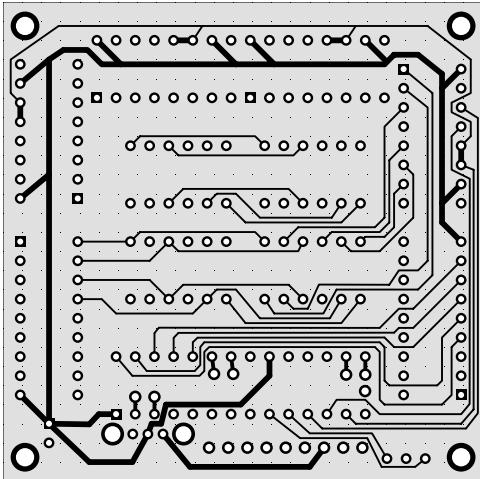
Appendix C: Hotpants Technical Documents

These documents describe the construction and operation of the Hotpants circuit. All of these documents and additional information is available at <http://acg.media.mit.edu/people/simong/hotpants/tech>.

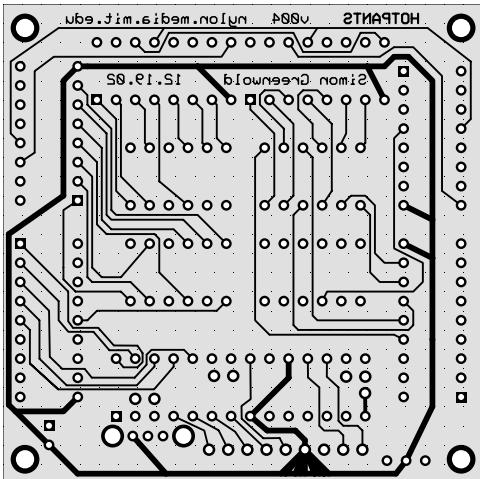


Component List

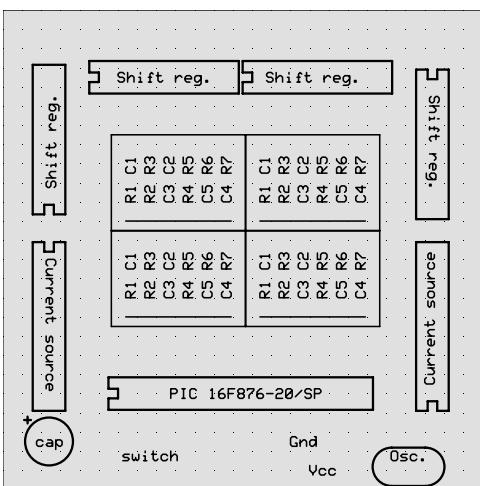
Component	Distributor	Part Number	Quantity per board
Shift register	Digikey	TC74HC595AP-ND	4
Oscillator	Digikey	X909-ND	1
LED Displays	Jameco	118906	4
Microcontroller	Digikey	PIC16F876-20/SP-ND	1
Capacitor	Digikey	P965-ND	1
Current source	Digikey	TD62783AP-ND	2
Switch	Digikey	EG1906-ND	1
Header	Digikey	A19343-ND	1
Battery Case	Digikey	BC4AAW-ND	1
AA Batteries	Digikey	P107-ND	4



Board front (actual size)



Board back (actual size)



Board silkscreen (actual size)

LittleVision firmware

This listing is the code that is burned into the microcontroller onboard the Hotpants to enable it to play movies. The movie data is loaded later with a bootloader (optional) starting at address 0x0396. Movie data is encoded with a change-marking compression. Assume the following states:

OFF = 0

HALF = 1

FULL = 2

We start each frame in state OFF. We advance from pixel to pixel, starting from the top left, and proceeding across each row of 14 down the display. We maintain a current global state. As we traverse the LEDs, we change the state of each LED to the global state. For each new LED, we read one bit of data. If it is a zero, we keep the current global state the same and simply set this LED to that. If it is a one, we look at the next bit. If that bit is zero, we add one to the current state (wrapping around to 0 after 2). If that bit is one, we add two to the global state. There are 14 bits per word in the PIC memory.

This code compiles using the PICC compiler from CCS [<http://www.ccsinfo.com>].

```

case
#include <16F876.H>
//#device *=16

// Configure PIC to use: HS clock, no Watchdog Timer,
// no code protection, enable Power Up Timer
//
#define DATA_START 0x0396
#define FRAMES_ADDR DATA_START
#define MOVIE_START (DATA_START+1)

#define USE BOOTLOADER
#ORG 0x1F00,0x1FFF //for the 8k 16F876/7
void loader() { }

// DATA RESERVE
#ORG DATA_START,0x07FF
void data0() {}
#ORG 0x800,0xFFFF
void data1() {}
#ORG 0x1000,0x17FF

```

```

void data2() {}
#ORG 0x1800,0x1EFF
void data3() {}

// Tell compiler clock is 20MHz. This is required for delay_ms()
// and for all serial I/O (such as printf(...)). These functions
// use software delay loops, so the compiler needs to know the
// processor speed.
//
#define DELAY(clock=20000000)

// Hardware definitions
#define OE      PIN_C1
#define DS      PIN_C0
#define CK      PIN_C2
#define C4      PIN_B7
#define C3      PIN_B6
#define C2      PIN_B3
#define C1      PIN_B4
#define CO      PIN_B5

// Hard parameters
#define W 10
#define H 14

#define SIG_DELAY 4 // Anything lower doesn't seem to work
#define REFRESH_PERIOD 7167 // Empirical. You can mess with it.

// Colors
#define OFF 0
#define HALF 1
#define ON 2

unsigned long movieWordPtr;
int movieWordLow;
int movieWordHigh;

#define STATUS=0x0003
#define EEDATA=0x010C
#define EEADR=0x010D
#define EEDATH=0x010E
#define EEADRH=0x010F
#define EECON1=0x018C
#define EEPROM 0x0007
#define RD 0x0000

//Movie data
#define FRAME_DELAY 15 // Determines the speed of the movie

unsigned int delayCounter;

int movieBitPtr;
unsigned long curMovieData;
unsigned int movieDataLow;
unsigned int movieDataHigh;

unsigned int nowState;
unsigned long curFrame;
unsigned long frames;

// Buffer storage
int frameCol;
short int inBufRefresh;
int bufNum;
unsigned long buf0[10];
unsigned long buf1[10];

void clearScreen() {
    int i;
    for (i = 0; i < 10; i++) {
        buf0[i] = 0L;
        buf1[i] = 0L;
    }
}

void point(int x, int y, int color) {
// int x, y;

    if (color == OFF) {
        buf0[x] &= ~(1L<<(long)y);
        buf1[x] &= ~(1L<<(long)y);
    }
    else if (color == HALF) {
        buf0[x] |= (1L<<(long)y);
        buf1[x] &= ~(1L<<(long)y);
    }
    else {
        buf0[x] |= (1L<<(long)y);
        buf1[x] |= (1L<<(long)y);
    }
}

buf1[x] |= (1L<<(long)y);
}

void getMovieWord() {

    movieWordLow = make8(movieWordPtr, 0);
    movieWordHigh = make8(movieWordPtr, 1);

    #ASM
    movf    movieWordLow, W
    movwf   EEADR
    movf    movieWordHigh, W
    movwf   EEADRH
    bsf     EECON1, EEPGD
    bsf     EECON1, RD
    nop
    nop
    movf    EEDATA, W
    movwf   movieDataLow
    movf    EEDATH, W
    movwf   movieDataHigh
    #ENDASM

    curMovieData = make16(movieDataHigh, movieDataLow);
}

void initMovie() {
    nowState = 0;
    curFrame = 0;
    movieBitPtr = 0;
    movieWordPtr = MOVIE_START;
    getMovieWord();
}

unsigned int getBits(int nBits) {
    int i;
    unsigned int result = 0;

    for (i = 0; i < nBits; i++) {
        result <<= 1;
        if (curMovieData & (1L << (13L - (long)movieBitPtr)))
            result++;
        movieBitPtr++;
        if (movieBitPtr > 13) {
            movieBitPtr = 0;
            movieWordPtr++;
            getMovieWord();
        }
    }
    return result;
}

void nextFrame() {
    int x, y;
    int nextBit;

    for (y = 0; y < H; y++) {
        for (x = 0; x < W; x++) {
            nextBit = getBits(1);
            if (nextBit) {
                nextBit = getBits(1);
                if (!nextbit) {
                    nowState = (nowState + 1) % 3;
                }
                else {
                    nowState = (nowState + 2) % 3;
                }
            }
            point(x, y, nowState);
        }
    }
    curFrame++;
    if (curFrame == frames) {
        initMovie();
    }
}

void runMovie() {
    while (1) {
        nextFrame();
        delay_ms(FRAME_DELAY);
    }
}

void clockSR() {
    output_low(CK);
}

```

```

delay_us(SIG_DELAY);
output_high(CK);
delay_us(SIG_DELAY);
}

void setupCols() {
    if (frameCol == 0) {
        output_high(C0);
    }
    else {
        output_low(C0);
    }

    if (frameCol == 1) {
        output_high(C1);
    }
    else {
        output_low(C1);
    }

    if (frameCol == 2) {
        output_high(C2);
    }
    else {
        output_low(C2);
    }

    if (frameCol == 3) {
        output_high(C3);
    }
    else {
        output_low(C3);
    }

    if (frameCol == 4) {
        output_high(C4);
    }
    else {
        output_low(C4);
    }

    output_low(OE); // oe
}

void drawFrameCol(unsigned long* buf) {
    int i;
    long mask;

    output_high(OE); // oe
    delay_us(SIG_DELAY);

    // TOP LEFT
    mask = 0b00000001000000;
    for (i = 0; i < 7; i++) {

        if ((buf[frameCol] & mask) != 0) // Light this row
            output_low(DS);
        else
            output_high(DS);
        delay_us(SIG_DELAY);

        clockSR();

        mask <= 1;
    }
    clockSR();

    // BOTTOM LEFT
    mask = 0b00000001000000;
    for (i = 0; i < 7; i++) {

        if ((buf[frameCol + 5] & mask) != 0) // Light this
row
            output_low(DS);
        else
            output_high(DS);
        delay_us(SIG_DELAY);

        clockSR();

        mask <= 1;
    }
    clockSR();

    // BOTTOM RIGHT
    mask = 0b0000000000000001;
    for (i = 0; i < 7; i++) {
        if ((buf[frameCol + 5] & mask) != 0) // Light
this row
            output_low(DS);
        else
            output_high(DS);
        delay_us(SIG_DELAY);

        clockSR();
    }
}

    output_low(DS);
else
    output_high(DS);
delay_us(SIG_DELAY);

clockSR();

mask <= 1;
}
clockSR();

// TOP RIGHT
mask = 0b0000000000000001;
for (i = 0; i < 7; i++) {

    if ((buf[frameCol] & mask) != 0) // Light this row
        output_low(DS);
    else
        output_high(DS);
    delay_us(SIG_DELAY);

    clockSR();

    mask <= 1;
}
clockSR();

clockSR();
setupCols();
}

void initTimer() {
    frameCol = 0;
    delayCounter = 0;
    CCP_1 = REFRESH_PERIOD;
    enable_interrupts(INT_CCP1);
    enable_interrupts(GLOBAL);
    setup_ccp1(CCP_COMPARE_RESET_TIMER);
    setup_timer_1(T1_INTERNAL);
}

inline void readNumFrames() {
    movieWordPtr = DATA_START;
    getMovieWord();
    frames = curMovieData;
}

void main() {
    clearScreen();
    readNumFrames();
    initMovie();
    initTimer();
    runMovie();
}

#INT_CCP1
void DRAW_COL() {
    if (frameCol == 4) {
        if (bufNum == 3)
            bufNum = 0;
        else
            bufNum++;
        frameCol = 0;
    }
    else
        frameCol++;

    if (bufNum == 0)
        drawFrameCol(buf0);
    else
        drawFrameCol(buf1);
}

```