Computationally Augmented Visual & Spatial Design

by Jocelyn C. Lin

Bachelor of Science in Computer Science and Engineering
2001
Massachusetts Institute of Technology

Submitted to the Department of Electrical Engineering and Computer Science on May 24, 2002 in Partial Fulfillment of the Requirements for the Degree of Master of Engineering in Electrical Engineering and Computer Science at the Massachusetts Institute of Technology

Copyright Jocelyn C. Lin, 2002. All rights reserved.

The author hereby grants to M.I.T. permission to reproduce and distribute publicly paper and electronic copies of this thesis and to grant others the right to do so.

Author
Department of Electrical Engineering and Computer Science
May 24, 2002

Certified by ... Peter Testa
Thesis Supervisor

Accepted by Arthur C. Smith
Chairman, Department Committee on Graduate Theses
Computationally Augmented Visual & Spatial Design

by Jocelyn C. Lin

Submitted to the Department of Electrical Engineering and Computer Science
on May 24, 2002
in Partial Fulfillment of the Requirements for the
Degree of Master of Engineering in Electrical Engineering and Computer Science
at the
Massachusetts Institute of Technology

abstract

Current design methodologies take advantage of the possibilities
of programmatic aids, but do so with an unfortunate rift between
the use of design and computation. This thesis explores the
possibilities of integrating computation into visual and spatial
design through a merging of processes and a combining of
programmer and designer. A series of experiments were con-
ducted in which computation played varying parts during the
creative process of deriving form, font, object, and space.

Thesis Supervisor: Peter Testa
Title: Associate Professor, Department of Architecture
acknowledgements

Professor Peter Testa, for his insightful guidance and open mind.

Devyn Weiser, for lending her design expertise, keeping me on track, and sharing Hello Kitty fun.

Ian Ferguson, for asking thoughtful questions and providing creative input.

Professor John Maeda, for giving a freshman a chance and inspiring words.

Mom, Dad, Jennie, & Jeffrey, for the support, trust, and encouragement through the years.

Steve Song, for being the best friend ever and for helping me out all the time.

Friends from home and MIT, for entertaining me and sticking by.

Pocky & Kiki, for being cute and looking like totoros.
# table of contents

1 introduction 11
  1.1 motivation
  1.2 background
  1.3 overview of thesis

2 experiments 15
  2.1 bikeTree 16
    2.1.1 computation
  2.2 Lux 21
    2.2.1 example
    2.2.2 computation
  2.4 Safety System 26
    2.4.1 computation
  2.3 Bumpyfont 30
    2.3.1 computation

3 evaluation 35
  3.1 process stage
  3.2 end design integration
  3.3 program vs. designer influence
  3.4 analysis

4 conclusions 39

references 41
list of figures

1.2.  
   a  GENR8 interface and product
   b  curvesculpt
   c  Individeo

2.1.  
   a  bikeTree stage 1 sketch
   b  bikeTree stage 2 sketch
   c  bikeTree video
   d  bikeTree wire sketch
   e  bikeTree code results
   f  bikeTree “4” model
   g  bikeTree “n” model

2.1.1.  
   a  bikeTree code run

2.2.  
   a  Post Office lighting study
   b  video analysis of activity
   c  Lux concept
   d  Lux office background differentiation

2.2.1.  
   a  Lux garage background differentiation
   b  garage floor plan and state diagram
   c  garage lighting and state diagram

2.3.  
   a  Everyware
   b  Safety stick sketches
   c  Safety System 3D applications
   d  Safety System 2D circle range
   e  Safety System 2D applications
   f  Safety System object modules

2.3.1  
   a  Safety System circle range
   b  Safety System visual cues
   c  Safety System hexagonal range

2.4.  
   a  Braille signage
   b  Pixelated fonts
   c  Use of program in Bumpyfont creation
   d  Bumpyfont variations
   e  buttons and handles
   f  doorframes

3.  
   a  evaluation chart
1 introduction

1.1 motivation

The designer today has an arsenal of computer tools at her fingertips. These programs can help to illustrate concepts, render structures, and animate characters. They have become an integral part of design within just the past few years. Almost every project an architect or designer works on is aided somehow by software.

However, this power of computation is often used as a tool for mundane tasks. This creates a divide between the use of programs and design, between the programmer and designer.

As it stands now, the design process is begun by a series of studies that help define a problem. Next, preliminary solutions are devised, analyzed, and refined. These actions cycle until a satisfactory answer has been reached. At any of these steps, a designer might use a computer program to help visualize an idea, but it is for enabling what has already been decided. The computer is brought in at the end of the idea generation, used in a disjoint act meant only to render the end product.

The rift between computation and design encompasses more than just the process. The separation between programmer and designer also helps in contributing to this problem. In the current situation, someone might see a need for a specific tool in design. The computer scientist studies the problem, then invests a large amount of time and effort in developing a complicated robust program. This program is then handed over to the designers who may or may not use it for its intended purpose, in the ways described in the previous paragraph. A programmer, after this point, has nothing more to do with his creation while the designers struggle to make it do what they want.

The design process is a creative one, as is the programming. There are many similarities and many ways in which one can supplement the other to create something new. This thesis was conceived in an effort to address this problem and to study how computation can be a more integral part of the design process. Hopefully, a new process can be derived combining designer and programmer while joining design methodologies and computational applications.
I would like to emphasize here that the problem I am studying is very specific, that of using computation to aid in the design of a product that might or might not be based on the technology used to create it. This means that the old model of creating general, usable, programmatic tools must be rethought since it is too complex a system to fit within a design cycle.

Towards this end, I devised a series of small experiments or design projects that would reveal different aspects of the problem. In each project, the entire design process was taken from conception to final renderings. I felt it was important to use this approach to follow the impact of computation in the design. Although some parts do not have an obvious relationship to the programmatic aspects, closer inspection might reveal some subtle yet important influence.

The potential of computation combined with design is much greater than simple pixel crunching. By harnessing this latent power one can arrive at a methodology in which design and computation are interdependent.

1.2 background

[fig. 1.2.a]
GENR8, programmed by Martin Hemberg, models created by Devy Weiser.

In the Emergent Design Group, an approach has been tried in using computation to explore various architectural problems. Through collaboration between architects and artificial intelligence experts, a series of software programs have been devised to address specific projects and problems in the design of spaces, materials and ALife [Testa & O’Reilly 1999]. Weaver is a program that enables multiple ways of weaving strands together. Various patterns and numbers of strands are combined and analyzed for their fitness for purposes such as cladding a surface [Testa & Greenwold 2001]. Another project, MoSS, and its cousin, GENR8, also deal with surfaces [Testa et al 2000, Hemberg et al 2001]. With these, the designer grows a surface based on a mathematical models (in this case, the Lindenmeyer
system and various interpretations). Parameters such as repellors and attractors are built in, which influence the evolution of surfaces. These collaborations lead to interesting aesthetics and structures that would not be possible without the input of both architect and programmer.

However, within the group, there is a distinct separation between the architects and the computer scientists. They first meet to state both of their interests and to think of what new things might arise from working together. Then, the programmers work on researching algorithms and developing new applications, meeting with the architects now and then to adjust the goals of the project. When done, the scientists relay the purpose and usage of the program and have little more to do with their creations. At this point, the architects use the program towards their own design ideas. Although the work is innovative, there is still that conflict of programmer versus designer.

In the MIT Media Lab, the designer and programmer are often integrated into the same person. The work there is very much based on new applications on technology, so that people working on projects tend to evolve a complex multimedia prototypical system. This is an interesting and often rewarding path to follow, but it is not one pertinent to this thesis. These systems sometimes have little practical value, and are often very time-consuming—a far cry from an attempt to stay within the bounds of the current design process and an attempt to design a more traditional product.

Other projects in the MIT Media Lab, again house the programmer and designer in the same body, but follow the program-design rift described above. In the Interactive Cinema Group, the Shareable Media system enables video collaboration. Individeo and Plusshorts [Seo 2001, Kelliher 2001] are two novel applications which were developed within the system, each by a single person. Both are video editors—an application to aid in creating an artistic vision. However the methodology used was again, the programmer-tool designer-create process. They spent a great deal of their time creating these programs and making them usable for a wide audience before creating their own videos to demonstrate the use.

[curvesculpt] is a piece I created while an undergraduate researcher with the Aesthetics and Computation Group. A curve shifts in response to mouse clicks, and collects past shapes into a three dimensional form. It is focused on interactions between user and program and does not attempt to solve a design problem.

[Individeo] is a collaborative video editor created by James Seo.
1.3 thesis overview

In Chapter 2, the design methodology of each project will be described, followed by a more in-depth look at the computational aspects of the study. Chapter 3 contains an evaluation of the experiments and the properties that make the approaches successful or unsuccessful. Finally, Chapter 4 will relate the results of this thesis.
2 experiments

In this section, the process of each experiment is related. Following each description is a more in-depth analysis of the computational aspect of the problem. The entire project is described to provide a context and to aid in evaluating the applications and effects of the programs. These programs are written to be easily adaptable, and not necessarily for robustness. This way of working allows for a more fluid process. Since the programmer and designer are the same, there is no need for a fancy user interface. With the purpose of the programs being experimentation, all projects steered away from a widely usable system.
2.1 bikeTree

The first study is a look into the methods involved in designing a bike rack, through the interactions between person, bike, lock, and rack.

To begin the process, I studied the forms of existing racks. Many of them are in reality repetitions of a two dimensional form. By experimenting with different incarnations of those forms, I found that the most visually appealing racks are those which borrow the shape of the bike frame.

The second stages involve looking at how the lock is utilized. Given that the only lock used is a normal u-lock for an average bike, certain volumes can be found that are optimal regions of locking. This also borrows from the form of the bike itself.

Another aspect to study is the effort involved in locking the bike. This extra work stems from several sources. One is the need to lift the bike at times with certain racks; another comes from the accessibility of the lock region. Since the bike is mostly two-dimensional itself, it can slide in and out of a slot easily, with the exception of the handlebars. These handlebars can interlock when bikes are locked too closely, causing more effort in disentangling.

Finally, a common complaint of bicyclists is that the act of locking the bike will often cause some amount of damage to the bike itself. By locking the rack instead of the bike, one can minimize this damage. Also, when doing this, the rack can be designed so that a more compact lock can be used.

This study into bike racks culminated in three different approaches. The first is an answer to different configurations, the second is a look into optimization, and the third is a more in-depth look at the lockable item.

In the configuration study, I wrote a program which can produce different rack forms based on different bike positions. Pulling from the studies before, the bike is represented as a 2D plane with two lockable volumes. I wrote the positions of each bike as well as the starting positions of the bike rack elements (rack-cubes) in the program itself. While running the program, each rack-cube branches out and creates children towards the lockable regions. At the end, all of the rack-cubes together form the shape of the rack, and each bike is lockable in some way from that rack shape. Less simple beginnings give birth to less obvious forms.
bikeTree code results

(fig. 2.1.e)
For the optimization aspect, I looked at the variables involved when using an average u-lock and a normal bike. The end result attempts to address each parameter and solve it efficiently. Each side of the '4' shaped element is made for locking either the front or the back of the bike. The diagonal portion allows for easy locking of the front wheel and frame, while the straight post lets the back wheel and frame be secured. By forcing bikes to orient in opposite directions, handlebars cannot lock, and bikes may be parked much more closely together. Also, the bikes can be simply wheeled in, precluding the need for any lifting.
Further exploring the idea of a lockable rack, I tried to lessen the material used while still having it support many bikes. One solution is the idea of a modular lockable piece which can fit around the frame or wheel of the bike. This piece is attached to a railing or some cylindrical rack form. The ‘n’ shaped lockable piece has the arched part attached, leaving the legs available for locking. This leads to the use of a much smaller lock that is easier for owners to carry around. By making it slide like a bead, the user can adjust the position so that it custom-fits her bike. Finally, there are stops along the rack so that the module is constrained between these points.
2.1.1 computation

The programmatic part of the bikeTree project is in finding different configurations of the bike rack. This is implemented in Java 3D. A three dimensional view is necessary for the work since the problem being explored is three dimensional in nature. Zoom, rotation, and translation are allowed for viewing this 3d world, but there is little other user-interaction. As mentioned before, these programs are not made to be particularly user friendly since programmer and designer are one and the same. Any changes desired can be achieved by tweaking a few variables, recompiling, and running the new program to view results.

The main objects in the bikeTree world are the bike itself and the rack cube. The bike is represented as a two dimensional plane with two rectangular volumes that approximate the locking regions.

The many rack cubes represent the final form of the bike rack. Each rack cube is generated from a predecessor rack cube which has branched. Connecting elements show how one is related to another as well as the progression of the bike rack from initial seed rack cubes. The program starts with an initial set of bikes oriented as desired and a few rack cubes placed in promising locations.

These small cubes spawn through a weighted chance algorithm. There is a 3x3 cube of possible positions for each rack cube children and the positions which are closer to a lockable bike region are weighed more favorably. Randomness is introduced to allow growth of less obvious solutions. On the whole, the tree of rack cubes grows towards the bikes, but might take some detours towards other bikes and show a form that is not immediately obvious.
2.2 Lux

Lux addresses the lack of responsive lighting. The idea originated from an analysis of office lighting – often the harsh overhead fluorescent bulbs are too dim or too bright for the worker and what suits one person may not suit another. People who use computers generally prefer a dark room to better see the monitor, whereas people who work with many papers need a lot of light. Even within a particular task, different levels of light can be required. A study for the US Postal Office looked at lighting for mail sorting [Mitchell et al, 2000]. It was determined that the people who had to sort mail benefited the most from a bi-level light system. The ambient light is supplied from a general light system, while brighter, more focused task lighting is furnished at each work station. This enables them to work with good visibility and a reduction of eye strain.

To find what sort of tasks someone performs in a particular setting, one could go through the tedium of following workers around an office and noting statistics. However, this could be circumvented by using computer vision techniques and algorithms. An interesting study by Kettmaker and Brand [2000] shows the possibilities of using video and computation to track a worker’s activities in his cubicle.

By applying this method to the problem at hand, one can find the pattern of space usage which allows the creation of a more interactive lighting.
The core idea of Lux then is to use computer vision techniques to track people within an environment, and using their behavioral states, plan a lighting system that can cooperate with the information. This lighting system is interactive in two ways. First, it is more finely tuned to the needs of those who use it. Second, it changes states of light based on what activity is occurring and what ambient light already exists. Transitions in behavior are tied to transitions in the lighting combinations.

At first, this was applied to a video of office workers. However, it turned out that lighting in the office needs to have more user control than this system allows, and furthermore, office behavior is not well suited for computer vision algorithms. The tracking algorithm used continuously updates the background model and depends on movement to find people. In an office environment, workers stay relatively still for long periods of time which effectively allows them to blend into the background. Other algorithms in computer vision, based on static background comparison would also fail when people move chairs around.

This was then applied to a parking garage situation, where personal control of light is not necessary and where constant movement provides an ideal testing ground for this method.
2.2.1 example

The first stage of the process requires sample video footage of an event, in this case, the Central Square Star Market garage usage. This was tested on sixteen minutes of usual usage patterns in the late evening, between 9:30 and 10:00pm. First, the footage is differentiated into foreground and background parts using multiple Gaussian filters to continuously model the background [Stauffer & Grimson, 2000]. Then, to label the foreground parts, a standard region finding algorithm is used [Sonka et al, 1999] which also aids in noise reduction.

a car enters the space. shown is the original video, the parsed pixels, and an overlay representation of the two.

in these overlays, one can see a shopper walking towards his car on the right, while another entering the supermarket disappears on the left.

After this initial processing, the screen is split up into parts corresponding to different locations, and a state diagram is derived.

the plan on the left shows the states for pedestrians.
- a1-2: entrance/exit to supermarket
- b1-3: walking junctions
- c1-4: parking or unloading groceries

on the right are the states for cars.
- d1-2: travel through loading lane
- e1-4: driving junctions
- f1-4: driving or parking
By logging the frequency of movement in a particular location, one can derive transitional probabilities from state to state. This leads to a behavior model for that space. Armed with this information, a lighting system can be devised which responds to that behavior.

[fig. 2.2.1.c] lighting, state diagram. In an actual application, the lighting system would be responsive to movement, which can be achieved by simple sensors hooked up to different lights.

below, a possible sequence illustrates lighting transitions for a car that enters, then parks, with a person walking from the car to the supermarket.

Ideally, this lighting system would also take into account the ambient light from the sun or moon. Unfortunately, with this particular garage, the heavy walls prevented significant sun penetration, and it was also brightly lit at all times of the day, which made it difficult to analyze the luminance.

Several things were discovered about the organization of that garage space. Most cars entered the area from the center, which happens to be the farthest from natural daylight. This resulted in more people just parking in that area for convenience. Such use meant that the most occupation was in an area that required the most light supplementation. A better design would
direct cars and pedestrians to the edges of the space to take advantage of the sun. Furthermore, on various trips to the garage, it was noted that the busiest time corresponded to the evening rush hour. Aligning the space to maximize light at these times would also save energy and improve the lighting.

2.2.2 computation

Programming was used throughout the whole project since the lighting was so dependant on the computational output. The JMF API was used to read and output video shot on a Sony digital camera. The video was cut into smaller segments and analyzed one at a time to reduce memory use, and the data was recombined later.

I extracted the pixel data from each frame of the video which is then sent through the Gaussian filters to differentiate foreground and background. This is done for each of the rgb values, then averaged. Foreground pixels are set to white, while black denotes background. These new pixels are then labeled into regions with the 8-neighbor 2-pass algorithm. Pixels which have an insufficient amount of neighbors are counted as noise and set to black.

Every sixty frames, the foreground piece (person or car) is then correlated to a location or region, and a regional counter is incremented. These counters can be used to derive a state diagram with transitional probabilities. They also relate the frequency of use in a particular area.
2.3 Safety System

One of the current projects in the Emergent Design Group is the study of distributed building systems, titled Everyware. Though an architect may design a space beautifully, the installation of electrical and plumbing systems often leaves much to be desired. Contractors fit items in a visually haphazard manner, relying only on regulations and standards, rather than an integration with the building itself. This applies to many systems that help a building run – the ventilation, plumbing, electricity, and safety. A very obvious example is that of the emergency or safety wiring. Exit signs hover in the middle of nowhere and pull alarms sit side by side, while the alarms they raise are tens of feet away.

In an attempt to better building design, EDG is pursuing the rethinking of many objects that make up these systems. In the emergency case, this takes the form of object integration. By combining multiple items into a one-stop station, named the “safety stick”, many benefits would be gained. There would be less materials used and building inhabitants would find and use these safety items more easily.
One approach to accomplishing this is to take the many rules surrounding the placement and distribution of these objects and create a new form based on them. Another is to base it on how they are connected by wiring and plumbing. However, the signal wiring aspect can be ignored since it is possible to create a wireless system that depends on radio frequency transmitters and receivers.

Several Java applications were created to help visualize the constraints and move around items within those ranges. These numbers were culled from several sources based on the NFPA and OSHA regulations [Beaty et al 2002, System Sensor 2002]. The first few versions were based on the “safety stick” form. For this, all of the items were represented in a three dimensional room, and displayed with a rectangular volume depicting the range of vertical movement. However, because their placements were so limited, most forms were vertical and columnar in nature.

The next few iterations of the application used their distribution constraints. Originally, it used circles to represent the range of use of each item, but since to adequately cover an area the circles must overlap, the circles were changed to hexagons.
After some attempts to place as many objects together as possible, it was evident that a “safety stick” type station was improbable and inefficient. The smoke detectors and sound alarms cannot be more than 40 feet apart while it is inefficient to have pull alarms spaced as closely as 40 feet.

Instead, the coding of the program and its application to various floor plans suggested a standard combination of items. After enabling the program to efficiently tile objects onto any floor plan by using a hexagonal range shape, various modules were defined.

There also needs to be three forms of these modules—one that is free-standing, one that is wall-mounted, and one that is corner-mounted. The corner mounted version is important since it negates the need for two wall-mounts in the visual items, such as the strobe alarm and exit signs.

[fig. 2.3.e] hexagonal plan evolution and different modules listed

<table>
<thead>
<tr>
<th>beginning modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>a* sprinkler</td>
</tr>
<tr>
<td>10’ radius</td>
</tr>
<tr>
<td>b* unit =</td>
</tr>
<tr>
<td>smoke detector / sound alarm / extinguisher</td>
</tr>
<tr>
<td>20’ radius</td>
</tr>
<tr>
<td>c* strobe lights</td>
</tr>
<tr>
<td>40’ radius</td>
</tr>
<tr>
<td>d* pull alarm / fire hose</td>
</tr>
<tr>
<td>80’ radius</td>
</tr>
<tr>
<td>e* exit sign</td>
</tr>
<tr>
<td>variable radius</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>end modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>18,000 sq ft office plan</td>
</tr>
<tr>
<td>06 exit signs</td>
</tr>
<tr>
<td>92 sprinklers</td>
</tr>
<tr>
<td>02 strobe</td>
</tr>
<tr>
<td>04 strobe + unit</td>
</tr>
<tr>
<td>02 strobe + hose/pull + unit-extinguisher</td>
</tr>
<tr>
<td>02 hose/pull + unit-extinguisher</td>
</tr>
<tr>
<td>24 unit</td>
</tr>
</tbody>
</table>
2.3.1 computation

The first version was created in Java3D and allowed the creation, movement, and rotation of items within their height placement ranges. This was displayed in a simple room which provided a local context for creation. It relied on mouse commands for viewing, and key commands for everything else. It was actually somewhat tedious to use and was quickly abandoned in favor of a more global view of objects.

The next iterations used an office floor plan space and shapes to represent a range of use. One could start out with “seed” objects and run a few generations to carpet the floor with them. Overlaps meant areas of inefficient placement while uncovered spaces were unsafe.

The circle version of this placed new objects and moved old objects based on their affinity with other types. This resulted in a rather haphazard look but led to the simplification of ranges to modular amounts of 20 foot multiples to encourage overlap and combinations of different objects. This was encoded into the hexagonal version. Here, the program placed each new object adjacent to an existing hexagon. If an object would be put just beyond a wall, the program nudged it back into the room. Efficiency was ensured with the use of a hexagonally shaped range. Walls were also accounted for, which led to a need for corner, wall, and free-standing forms. Doing so also helped to depict where visual items needed to be placed.

![fig. 2.3.1a](image)
circle's affinity placement with messy visuals

![fig. 2.3.1.b](image)
blank regions indicate that more exit signs are needed

![fig. 2.3.1.c](image)generations run on hexagonal range floor plan

seed items are placed where required, in this case, by the exits

the program then tiles the items to fill the space, adjusting positions as needed

gaps are easily filled in by hand afterwards. inefficiencies appear as “cracks” of non-filled space in the middle and the overlapping regions at the ends of the room. this suggests that the floor plan could be changed to accomodate these safety objects better.
2.4 Bumpyfont

During the study of building systems, I became aware that the signage within the building is often inefficient, redundant, and bloated. A prime example of this is the plaque beside a door denoting its purpose. Braille for the blind and text for the sighted are separated which doubles the space needed to convey the same piece of information. Signage is often not put in the best location, such as in the case of the elevator buttons. There is a button to push, then a plaque beside it relating the function. This recurs in many situations.

Bumpyfont is a merging of visual and tactile information intended to address this problem. By superimposing typography and Braille, a new font can arise which is truly dedicated to its function of conveying information, and does so in a compact and meaningful way.

Another influence in the creation of Bumpyfont is the abundance of pixilated fonts existing – from LED displays to novelty type. It is not a far step then to imagine Braille as a significant component of this sort of type.

After this initial study and identification of the signage problem, I made an application to help create the font. The program provides a gridded canvas to paint on new letterforms and allows the saving and loading of created fonts. Within each grid, a non-changeable pattern of Braille exists which correlates to the letter being formed. This is the key purpose of the program – to help keep in mind the constraints from Braille since the common letterform is already familiar.

I chose to use the lower case alphabet because in Braille, uppercase requires the addition of a symbol to denote the difference. Since the main purpose of Bumpyfont is for signage and not mass amounts of text, lowercase makes the spacing more efficient while being more legible. Also included are all of the numbers and a few of the punctuation marks that may appear in signs.

I created different fonts on different versions of the application. While working, I constantly designed and revised to address different factors which came up. With each version, the resolution became finer – each lettergrid contained more cells. (Here, cells do not refer to the Braille convention of an entire 6 dot letter, but instead, represent the small blocks within each letter's grid in the program.) At first, I had compensated for the different widths of 'm' and 'w' by making a grid with wider dimensions, but later decided on uniformity and compactness in the font and changed the grids back to identical widths. After a
while of tedious cell filling by hand, I set the grids to automatically contain the two vertical strokes which overlap the Braille columns. This saved much effort while allowing more subtle alterations to each character.

[fig. 2.4.c]
a font in progress

< the creation panel provides gridded letters to paint in

< the end of the second row has not been completed yet & still contains the programmed double stroke

< here, the input panel & the display panel are used in tandem to check on similar letterforms

[fig. 2.4.d]
Bumpyfont variations and corresponding characters in Adobe Garamond
The applications of Bumpyfont are meant to be small and interactive. Bumpyfont exists where people naturally touch an object and conveys appropriate information. When applied to an object, it serves an instructional purpose. Commands such as push, pull, or turn could reside on door handles. ‘Press’ would exist on water fountain buttons and handicap door push buttons. Elevator buttons would have floor numbers and commands on the button itself, instead of on a nearby plaque.

Bumpyfont would also be useful as an identification tool. When placed on doorframes and moulding, they can relate the function and location of the door or the space beyond. Some examples of common terms are shown in the figure below. If applied systematically, it helps to regulate for both the visually enabled and impaired what purpose otherwise identical doors may serve. They could be applied in several ways. The text could be worked into the material, made into laminated stickers and applied, or stamped onto plaques as they are currently.
exjd

elevator

doors

[fig. 2.4.d]
doorframes and mouldings

restroom

men

women

office

auditorium

26.100

26.100
Finally, this new font could further the cause of Braille itself. By placing the two in obvious correlation, those who are sighted are encouraged to learn basic Braille and gain more understanding of the visually impaired. Another group of people who may benefit are those who have such reduced vision that the use of Braille is necessary. This sometimes occurs through a gradual loss of vision, so that it is difficult and frustrating to have to depend on Braille when some vision still exists. By using Bumpyfont, perhaps the transition can be eased and the comprehension speed can be increased.

2.4.1 computation

The code is organized by purpose – each panel corresponds to a different use. The creation panel allows for easy manipulation. Each letter is represented by a grid of cells, some of which are permanently filled in to represent Braille bumps. The dimensions of the bump and the spacing in between are easily changed programmatically. The resolution is also designed to be easily altered in both the pixel dimension of a cell, and the cell dimension of a grid. Being the programmer designer aided in deciding what dimensions letters ought to be, and allowed for quick changes, such as in the 'm' and 'w' example mentioned before. Also as mentioned before, it was relatively painless to make lettergrids pre-filled with two vertical strokes since they are common to many lowercase letters. This neat correspondence between Braille columns and letter strokes was in fact another one of the inspirations for this project.

Next, display and input panels were added to give better feedback. They aid in creating letters. One can type any phrase in the input box and see what the currently edited font will display it as. This allows for easy comparison between letters that are meant to look similar, such as 'p'/'q' and 'h'/'n'. This also helps to visualize the form of an entire word and promote legibility. The display panel was easily adapted from the creation panel by copying the lettergrids over, removing the gridding, and shrinking to size.

Finally, file save and load were added. Being able to save work for later perusal is an obvious need. This also helped in creating new fonts since styles could be modified from old fonts and saved as a new one. This method of working is similar to that of the prefilled lettergrids, where programming helps save effort when the design is known.
3 evaluation

With such a disparate group of experiments, it is difficult to determine the success without any sort of standards. Towards that end, I have attempted to distill some main categories in which the design-computation interaction can be evaluated. Each project is compared and discussed in those terms.

These different categories are as follows. The first is an evaluation of the process stage and the portion where computation was incorporated. Next is the end design integration, the influence of computation on the end result. Finally, the impact of the designer versus the programmer on the project itself, and how well computation was used is discussed. Concluding this section is an overview and analysis of all projects.

3.1 process stage

The experiments conducted varied widely in which portion of the process contained programmatic elements. Both Lux and Bumpyfont included computation in the middle designing stages, after all the preliminary studies had been finished. In contrast, bikeTree used computation before many parameters had been set, and was used to help determine possible future forms. This often led to shapes that were too simple and obvious, which could have been derived without the aid of computation. Safety System was also used to help find possible forms, but the programmatic aspect actually came much later — the elements in the project had already been very much refined and distilled. There was already a firm idea of the structure the project would take and how computation would fit in.

3.2 end design integration

On the whole, most of the projects had a notable influence on the end design. Since the letterforms of Bumpyfont were
defined by the constraints imposed by the program, it was influenced in shape even though the end form was very physical. Conversely, the final context and use of Bumpyfont were not in any way affected by the program. Safety System also depended on the computational aspect—certain forms and groupings of objects were derived from the application which would have been otherwise difficult to discover. Towards the far end of the spectrum, Lux was fully integrated with the code since it was required for determining movements and patterns.

On the other end, bikeTree did not seem to have significant input towards a final design. As a form generator, it provided an interesting view of possibilities but did not contribute new ideas. This may be due to various sources. Either the computation did not provide novel enough forms, or the project had not been pursued to a great enough depth to allow a good application.

3.3 computation vs. designer influence

This section evaluates how much the use of computation was affected by being a designer versus being a programmer. In bikeTree, the studies beforehand led to the program being structured around the bike planes. However, once these preliminaries were set, the program followed its own rules to generate forms. Any manipulation of data and placements occurred within the program—one could go back and change the values of some variables to show different things. So, the forms that were output were affected by design studies but more influence was felt from the programming. Lux was much more computationally based. The processing of video was entirely dependent on various algorithms and the designing did not take place until after the information was gathered. The state diagrams generated became a part of the design afterwards.

Bumpyfont and Safety Systems were very reliant on constant changes and adaptations due to both designer and programmer. In Bumpyfont, the patterns of Braille led to an easily adapted, rigidly structured application. Throughout the process, shapes dictated the addition of new code while the code dictated letter outcomes. These were only constraints though, and the actual formation with the “painting” of the letters stemmed from more of a designer-tool aspect. For Safety Systems, a good deal of design research was conducted before the programming began. Thus, the program was more of a sketching method and a generative application from which results were
viewed. The outcome defined the design possibilities of form, and also informed the successive versions of the same application.

3.4 analysis

In looking back on the completed projects, the more successful ones have several factors in common. Generally, the ones which include computation at a middle to later stage in the design process result in a more meaningful dialogue between programming and design. The attempt at integration of computation into the design process needs a definite timeline. This means that the study of the problem must be solidified enough to do so. It should be narrowed down enough so that a relatively structured program can be creative enough to contribute something. The early stages of the process should be avoided since there is too much flux, and idea generation is too quick to allow the relatively slower building of computation aids.

Another time based issue that occurred was the attempt at correlating the time scales of design and programming. Early design tends to be a constantly changing, quick moving process. When compared to the initial time required to create a structured program, it is difficult to justify the time expense. So, again, the design process should be at a point where a program can serve to sketch and generate ideas.

It was also difficult to allow the program to contribute something unachievable otherwise because of the time constraint. bikeTree tended to yield simple forms, but Bumpyfont circumvented it by allow complexity to arise from the designer-user. Lux used the computational power to accumulate information. Likewise, Safety System took advantage of the generative, number crunching ability and the adaptability in floor plans to be useful. So, a few different methods worked in accomplishing a significant programmatic contribution. To make it meaningful, computation was used for an analysis of many elements, as a sketching medium, as an adaptable multi-situational application, or as a means to encode constraints.

These can be achieved from having computation as a central part of a project, as in Lux, or by using it as a means to an end. The amount of end design integration and influence is largely dependant on the fluidity between designer and programmer. It is a success when the design owes as much to the program as the program to the design.

An unaddressed problem was that of judging when to stop programming in order to focus more on the design of the object
itself. This recurred through the projects and was particularly relevant if the final form was not to be computational. As in design, programs can be refined ad infinitum, and it was difficult to decide when to stop developing further and work on actualizing the product. In the future, this would be a worthwhile aspect to experiment with to refine the working process.
Through working on these various projects, I have found a method that works for me in using computation while designing. This computation is meant to fit into the current methodology of design, constrained by the current time requirements. It is used as a means to an end – it could be integral to the product or show its influence in more subtle ways. When attempting to include computation, the problem should be well defined. That means all preliminary research in design is best achieved in traditional manners – not only will it lead to more meaningful applications of computers, but for such a wide range of factors, it is faster and more useful to think of possibilities that way. Once this early study has been completed, it is easier to establish what can be done to aid the design.

To take advantage of the programmatic power, ideal applications would be constantly changing sketching tools used to analyze, generate, and constrain ideas. To do so quickly, it is easier to violate some rules of “good” programming. Since the only user is also the programmer, the interface is the code itself. This allows for any change, from simple parameters while designing to a shift in project focus.

Each project requires a new approach and an individualized application. General solutions are not as feasible since what makes them general is the complexity and robustness of the system. Such applications present the problem of creating and maintaining a time-consuming project.

Later, it may be interesting to apply this approach in a team of programmer-designers to see what solutions might arise. Perhaps with more people, the time expense of creating programmatic aids would not be so great. However, care must be taken to stay away from creating a massively complex code that’s unusable by everyone.

This thesis has approached the problem of the design-program rift and has begun to find a methodology which works for me. Although none of the projects were every fully carried to product completion, they showed that the computational medium is a useful and feasible one in the design process, and when linked in meaningfully, can contribute a great deal.
references


All images in this thesis are created by the author, unless otherwise noted.