CHAPTER 1
WHY BUILD A DIGITAL SANDBOX?

1.1 MOTIVATION

Earth-forming – the act of manipulating the surface of the earth in three dimensions – is one of the most important and fundamental tasks of landscape architecture. It is an integrative act that requires a designer to have a comprehensive understanding of the structure and function of the land, from its ecological dynamics to its aesthetic qualities. As a result, landscape architects have multiple goals in mind when involved in earth-forming, and to meet these goals, they typically use a variety of tools – from pen-and-paper sketches to heuristic ecological models.

While many of these tools are useful – even indispensable – to designers, they also limit the creativity and flexibility of the earth-forming process. Some tools provide more support for creative design tasks, while others focus more on quantitative analysis. For example, traditional analog tools such as sketches are used by designers to explore the aesthetic and spatial qualities of a design. Cardboard models further support design by enabling a landscape architect to explore a design in three dimensions. Neither tool allows a designer to quickly and accurately analyze the effect of a design on an ecological process like stormwater runoff. Analysis of this sort is well-supported by existing digital analytical tools like geographic information systems (GIS); however, the majority of these tools do not provide the kind of support for aesthetic, three-dimensional
design exploration that analog tools do. More importantly, no tool enables the landscape architect to combine these design and analysis tasks in the same work environment.

1.2 The Problem

Consider a simple hypothetical scenario to illustrate the point. In a typical design project for a neighborhood park, a landscape architect might be given the following objectives:

Develop a small neighborhood park to serve two primary purposes:

1. The park should be a focal point and gathering place for the community.
2. The park should provide ecological infrastructure to filter and store stormwater runoff from the adjacent neighborhood.

The site is located in an urban context and is directly adjacent to a stream corridor. Although the stream is far from “pristine” condition, nearby reaches have been targeted for restoration. Thus, it is important for potential designs to filter stormwater runoff and release it into the stream at a controlled rate, while integrating this ecological process into the pedestrian experience of the park.

The project requires the designer to juxtapose two seemingly contradictory concerns: to create a compelling aesthetic experience while simultaneously controlling stormwater runoff. Earth-forming will likely play a critical role in the landscape architect’s final design, since topographic manipulation is a fundamental technique for site design. How will the landscape architect proceed?

The typical designer might begin the project by representing the existing topography of the site using a variety of tools. His starting point is a contour map, on which he quickly diagrams the site’s characteristics: low points and high points, ridges and valleys, site lines and viewsheds. The designer uses the contour map to create a three-dimensional model of the site in cardboard. Together, the map and model enable
him to visualize the volume and form of the land, and he uses these representations as an empirical baseline for the design work to come.

The designer next begins exploring design alternatives by illustrating his concepts in a series of line drawings. He depicts critical changes in topography by sketching sections through the most important portions of the site. He develops a sequential spacing of earth forms by drawing new contour lines in plan view. Once he is satisfied with the overall form of the design, he builds a cardboard model to explore the design in three dimensions. To create an aesthetically compelling experience, the designer works back and forth between the different media as he artistically sculpts the land.

The designer also needs to determine how the design will influence patterns and volumes of stormwater runoff across the site. To estimate where stormwater runoff will accumulate, he traces flow lines as best he can from the contour lines of the proposed site plan; he measures slope angles and readjusts contours as needed. He will use these flow paths to direct stormwater to a wetland that will be designed and sized to filter and store the water. His design process becomes more deliberate now, as he consciously manipulates contours to direct and filter the expected runoff. If he is ambitious, he digitizes the contour map and uses a GIS to predict water quantity and flow paths. It is more likely that he will simply trace flow lines and measure drainage areas directly from the contour plan. When he is finally satisfied with his creation he draws up the construction documents, and the design work comes to a close.

This hypothetical scenario illustrates several important points about the process of earth-forming and the tools used to support it. Landscape architects have multiple goals
in mind when engaged in earth-forming projects, and they typically use different tools to accomplish different tasks. In the example above, the designer used the standard cadre of earth-forming tools – line drawings, physical models, and digital or quantitative models – to represent, express, and analyze the design. These various tools are used at different stages of the design process and for distinct purposes: pen and paper for quick study sketches, models for three-dimensional exploration, and heuristic analytical tools to model ecological processes. Each technique is valuable at certain stages of the project, as it provides different types of information about the design. Each also has its own set of disadvantages, since no single tool provides all the information a designer might need throughout the entire design process. More importantly, none permits the landscape architect to move seamlessly between design and analysis. Designing is done in one tool, analysis in another. The landscape architect is forced to interrupt the earth-forming process and “transfer” the design – conceptually and physically – between different media.

1.3 ADVANTAGES AND CONSTRAINTS OF CURRENT EARTH-FORMING TOOLS

Each tool used in the earth-forming process has its own advantages and constraints. The most widely used tool is probably the sketch. Designers use sketches to represent topography either by outlining contours in plan and section or by drawing orthographic projections (Strom and Nathan 1998:1-9). Contours are two-dimensional lines that represent equal elevations along their entire length. They are a standard method for representing topography on maps, particularly when other information needs to be
displayed simultaneously without visual clutter. Designers typically make sketches of contour lines when manipulating the elevation of a site (Figure 1). Because contour lines provide relatively accurate representations of topography, they are useful in specifying the vertical relationships between site elements. But there are disadvantages to using contour lines (Petschek 1996). They are only a two-dimensional representation and require the designer to constantly conduct internal visualizations to mentally extrude the contour map into a three-dimensional form. Thus, contour lines can be awkward in the early phases of conceptual design. Moreover, editing contours is labor-intensive and tedious. Local changes in one area of a site often have ripple effects outward, and these...
effects require the designer to rework and recheck contours across the site (Westort 1998:25).

Designers also use sketches to make orthographic projections and perspective drawings (Robbins 1994:23). Because they give the illusion of three dimensions, these types of sketch help a designer visualize the shape of a landform (Figure 2). Sketches

Figure 2. Earthwork sketch, Andrew Goldsworthy. From Friedman and Goldsworthy (1990:132).
provide designers with a quick way to illustrate and clarify ideas and are thus more helpful than contour-line drawings in the early phases of the design process. Landscape architects and environmental artists alike use sketches to develop design ideas (e.g., Causey 1990:134; Beardsley 1977:90). Regardless of its utility, the primary advantage of the orthographic sketch is also its main drawback – it only gives the illusion of three dimensions. If a designer would like to see a site from a different angle, a new sketch has to be made. This process can be just as time consuming as redrawing contour lines. Neither tool supports conceptual, three-dimensional design exploration as well as it could.

Physical models give designers a better understanding of the three-dimensional form of the land than do sketches (Figure 3). Models, whether digital or physical, are

Figure 3. Physical model of a landform, Oosterhuisassociates. From Zellner (1999:74).
typically developed directly from the contour lines of a site plan. In contrast to contour maps, however, no mental extrusion is necessary to visualize the volume of the landform – the designer can simply walk around the model to view it from an infinite number of perspectives, without having to remake it each time. But models are time consuming to construct. Cardboard and clay models can take hours to build, and if a design changes, the process of editing a physical model is very labor intensive. Often an entirely new model will need to be constructed. This, after a new contour map is drawn to act as a base for the model. As a result of model-building’s intensity, more time is spent building the model than crafting and editing the actual design.

Digital models are a different type of tool altogether and enable forms of representation and analysis that are distinct from those made possible by analog media (Ervin 1994). Most digital models represent topography using an array of points – typically x-, y-, and z-coordinates – stored in a file. These spatial coordinates are used by computer-aided design (CAD) software to construct a digital topographic surface that represents the three-dimensional shape of a landform (Figure 4). Geographic information systems combine these spatial coordinates with other attributes of the terrain – like land cover or soil type – to permit a designer to create quantitative models that are useful for computationally intensive tasks such as ecological modeling. Three-dimensional digital models resemble their analog counterparts in that a designer is able to interactively navigate through the site. Like their physical counterparts, digital models are usually constructed in a medium that requires a designer to concentrate on editing tasks rather than on design tasks. The same is true of designing and running ecological models that
are used to quantify phenomena like stormwater runoff or landscape ecological structure. As a result, the process of analyzing the ecological impacts of a design is time-consuming, and designers are thus less likely to conduct such analyses. Even though digital modeling tools offer advantages that traditional analog tools lack, they are often less likely to be used in the earth-forming process.

Although the designer in the hypothetical scenario uses a combination of standard tools, none of them meets his needs at every point in the design process. Instead, the process is interrupted because design is disconnected from analysis. The connection between the two is critical, since analyzing the effects of a design allows a designer to reflect in action – to consider the multiple connections between the site elements being manipulated (Schön 1985:51). If, as Simon (1981) suggests, design is a decision-making process that consists of generating alternatives, and then testing these alternatives
against a set of preconceived requirements, then it is advantageous for earth-forming
tools to give the designer the ability to make decisions that are crucial to crafting a good
design and meeting the project’s requirements. In light of the seemingly contradictory
goals of the above scenario, it would be more helpful if the designer had a single tool that
could be used both to sketch and to compute, to sculpt and to quantify – thereby
integrating design and analysis in the solution of earth-forming problems.

1.4 GOAL OF THIS STUDY

The goal of this study is to design and build a digital computational tool that
provides better support for earth-forming. This new tool explicitly addresses the
separation of design and analysis that exists in most earth-forming tools by giving a
designer the ability to design a landform and simultaneously analyze the flow of
stormwater over that landform. It does so by attaching a quantitative model of an
ecological process to a spatial model of a landform. In addition, it marries this combined
model with a gesture-based modeling interface that allows a designer to intuitively design
the spatial model using hand gestures in three-dimensional space.

The name of the tool – the Digital Sandbox – captures the two most important
aspects of the tool. I use the sandbox metaphor to express the intuitive, gestural interface
of the tool and the way that it approximates the working environment of a traditional,
physical sandbox. Although the sandbox is not a staple earth-forming tool of most
landscape architects, it is one of the few media in which a designer can directly and easily
express a three-dimensional concept. Because sandbox models are so easy to build and
edit, they reduce the lag between design conception and spatial expression. The digital
implementation of the tool enables it to transcend some of the limitations of a traditional sandbox, like the inability to serve as a platform for the computation of natural processes. By embedding an ecological model within the sandbox itself, the Digital Sandbox begins to bridge the divide between design and analysis in the earth-forming environment.

1.5 Methodology

I approached this study in two ways. First, I conducted a review of related work in the fields of design computing and landscape architecture to explore precedents for a digital sandbox. I was particularly interested in the intersection of three topics: 1) methods for digital earth-forming; 2) gesture-based methods for digital modeling; and 3) methods for integrating design and analysis in digital computational tools. The outcome of this review provided the motivation and rationale for designing a digital sandbox. Secondly, I developed the Digital Sandbox by extending a gesture-modeling system to support three-dimensional earth-forming and ecological analysis. The resulting work environment allows a designer to gesturally create a digital landscape and then run a simple stormwater accumulation model on the landscape.

1.6 Structure of this Document

Chapter 2 presents a review of design computing work related to this project. Chapter 3 provides an overview of the Digital Sandbox implementation, as well as an illustration of its use in a typical design scenario. Chapter 4 summarizes the project, draws a number of conclusions about the successes and shortcomings of the Digital
Sandbox, and speculates about the future of earth-forming tools and their potential impact on landscape design.
CHAPTER 2
PRECEDENTS AND RELATED WORK

2.1 INTRODUCTION

The concept for a digital sandbox is situated at the intersection of three issues in design computing: methods for digital earth-forming, gesture-based methods for digital modeling, and methods for integrating design and analysis tasks in digital computational environments. Tools in all three of these categories contain advantages and shortcomings that together provide the motivation and rationale for the design of the Digital Sandbox. In this chapter I review related projects in each of the categories to provide historic context for my project, to identify conceptual issues the projects seek to address, and to suggest methods for designing better earth-forming tools.

2.2 METHODS FOR DIGITAL EARTH-FORMING

Although earth-forming is fundamental to landscape architecture, there have been surprisingly few innovations in the digital tools used to accomplish it. The inherently complicated nature of terrain, and its representations in the digital realm, have frustrated attempts to develop new tools for topographic manipulation (Sawyer 1998). But there has been some progress. The Topographic Surface Sculptor (TSS) is a digital earth-forming tool that both mimics and transcends the abilities of a traditional bulldozer (Westort 1996). The TSS consists of three parts: a blade tool for sculpting paths across a digital surface; a library of geometric primitives used to construct a digital landform; and
boolean and algebraic operations to combine the geometric primitives into a single landform (Figure 5). To create a digital landform, the user selects a bulldozer blade and adjusts its shape, defines a path along which the digital surface will be transformed, and then applies the transformation to the digital surface. When the transformation is applied, TSS deforms the surface along the specified path in the shape selected by the user. The vertical dimensions of the deformation are controlled by adjusting the shape of the blade and the horizontal dimensions are controlled by defining the path.

Several conceptual issues informed the design of TSS (Westort 1998). First, Westort strove to make digital earth-forming qualitatively better for landscape designers...
by incorporating concepts from traditional sculpture. The motivation was to develop a
generic digital sculpting tool and explore its implementation in the TSS. Secondly,
because she recognized that designers typically spend too much time working with the
medium rather than with a design, Westort was concerned about separating “editing”
tasks from “analysis” tasks in the earth-forming process. The TSS attempts to focus the
user’s attention on the design being created, rather than on the medium, by using a simple
blade-and-path tool to maximize geometric control over the digital topographic surface.
As a result, the TSS alleviates some of the awkwardness associated with existing earth-
forming tools.

A different approach is taken in Leveller™, a commercial heightfield modeler
developed by Daylon Graphics (Daylon Graphics 2001). Leveller™ is a raster-based
paint program that uses color to represent elevation across a gridded surface of pixels
(Figure 6). The software allows the user to edit the heightfield using methods similar to

![Figure 6. The Leveller™ interface: 3D view (left) and 2D editing window (right). From Daylon Graphics (2001).](image)
those a conventional paint program uses to edit bitmap images. Editing the heightfield is
accomplished through a suite of tools that enables the user to sculpt it and give it three-
dimensional shape. Similar to the TSS, Leveller™ also contains a set of tools that
deforms the digital surface according to a variety of shapes. For example, “dig” and
“raise” tools excavate and add terrain, while the “flatten” tool creates areas of equal
height. The user can select various brush sizes to exercise more control over the shape
and extent of these deformations. Although all editing is done in plan view, a separate
window provides the user with a perspective view of the heightfield, enabling changes to
be immediately visualized in three dimensions.

The TSS and Leveller™ are improvements over existing CAD-based tools, where
mesh landform models are created from digitized contours in a tedious and time-
consuming process. Both systems utilize procedural metaphors that provide better
support for creative earth-forming, especially in the early conceptual stages of a design.
The “paint” metaphor used by Leveller™ is no doubt comfortable to designers familiar
with image-editing software, while the blade-and-path concept employed by the TSS
mimics the processes used in actual landform construction. More importantly, although it
is unlikely that any landscape architect has used a bulldozer in the early conceptual stages
of an earth-forming project, the TSS makes it possible for a designer to comprehend the
physical constraints of earth-forming by more closely matching the design tool to the
construction process.
2.3 GESTURE-BASED METHODS FOR DIGITAL MODELING

To support digital earth-forming, the Digital Sandbox utilizes a gesture-based interface instead of a traditional graphic user interface (GUI). Although GUIs have been the basis for nearly all advances in computer software over the past two decades, they require designers to abandon the familiar methods of interaction they use when working with traditional, physical tools like sandboxes. Gesture is one of these methods. It provides a rich and expressive medium that can communicate intent not easily conveyed by traditional GUI interfaces. Designers use hand gestures in sandboxes to directly and easily create three-dimensional landforms. The combination of natural gesture and pliable medium serves to reduce the lag between design conception and spatial expression, which makes the sandbox an excellent tool for earth-forming. Because gesture is important to sandbox sculpting, gestural interfaces have considerable potential for extending this direct and natural method into the digital realm.

In recent years, a number of gesture-based interfaces with explicit support for design applications have emerged. One such interface is GestureVR, a vision-based, three-dimensional gesture-recognition system (Segen and Kumar 1998). GestureVR is implemented in a desk-top environment and uses optic input technology. Two video cameras are placed about one meter above the gesture space to capture the plan and profile view of the user’s hand, thereby eliminating the need for the user to wear an instrumented glove. The system captures and describes the spatial location and configuration of the user’s hand by computing several input parameters: the x-, y-, and z-location of the fingertip and the azimuth and elevation angles of the finger’s axis. In
order to keep the analysis robust, the system makes use of figure-ground contrast. The
gesture space is framed by a background of uniform color and intensity, and the contrast
between this background and the user’s hand enables the two to be reliably separated.
Although somewhat inconvenient, this method makes for a fast and reliable recognition
system.

GestureVR was developed as a generic input interface to be used with any type of
application. To demonstrate its efficacy in CAD applications, the authors developed a
specialized implementation they call “3D Scene Composer” (Segen and Kumar
1998:457). In the Composer, the user can incrementally build three-dimensional models
by creating and manipulating digital objects (Figure 7). The user can draw lines and

Figure 7. GestureVR interface and 3D Scene Composer. From Segen and Kumar
(1998:Figure 1).
curves by tracing them with an index finger, select objects from an on-screen palette and put them in the three-dimensional scene, reposition the objects, and rotate the scene. Because the system reads input for only one hand and each gesture is directly mapped to a command, GestureVR is robust and was easy to implement.

Other gesture-based systems have attempted to transcend the limitations posed by the single-handed interaction of GestureVR. For example, Nishino et. al. (1997) developed a gestural interface they call the Two-handed Gesture Environment Shell (TGSH), a unified software framework that effectively utilizes two-handed gestures. The system acts as a generic interface between the user and various virtual reality applications. The TGSH is not vision-based like GestureVR. Instead, it uses instrumented gloves to communicate with the virtual environment. A driver reads the data stream from the gloves to determine the shape, position, and orientation of the user’s hands. Dynamic hand gestures are recognized and processed by a recurrent neural network in a pattern-matching process that is much more intricate than that used by GestureVR. Most importantly, the TGSH was found to be more stable and efficient when both hands were used. Two-handed gestures significantly increased the amount of information available for pattern-matching on complex gestures and provided greater expressive power for the user by increasing the number of gestures available for use (Nishino et. al. 1997:6).

One of the shortcomings of gestural interfaces like TGSH and GestureVR is their one-to-one mapping of gesture to command. According to Wexelblat (1995), such a strategy reduces the expressiveness and idiosyncrasy that is inherent to gesture. As an
alternative, he proposes a more sophisticated and generic approach to gesture modeling. His system is a continuous-gesture recognizer and takes the concept of gesticulation – the ordinary form of gesture that people use in everyday situations – as its point of departure. The project’s goal is to understand and encapsulate gesticulation so that it can be incorporated into any computing environment as a generic, independent interface.

Wexelblat’s system decomposes gesture recognition into two discrete but related stages: *analysis*, where features are detected, and *interpretation*, where meaning is assigned to the features (Wexelblat 1995:186). The analyzer produces an intermediate representation of a gesture, while the interpreter uses this representation, along with the user’s environment, to assign final meaning to the gesture. Separating gesture recognition into these two components enables gestural input to be used in any context. As a result, the same gesture can have different meanings, depending on the context in which it is used.

In Wexelblat’s prototype system, sensors are placed at various points on the user’s hands and upper body to create a model of the user’s body. The application is a simple room layout system that enables users to create and move furniture in a simulated room. The analyzer receives input from the body model and communicates this input to the interpreter. The interpreter then uses gestural input to model the virtual room and to transform the room’s objects. For example, if the user says “put a chair there” and gesturally indicates a path along which the chair should be moved, the analyzer captures the path information and passes it to the interpreter. The interpreter then uses the information, along with data from the speech recognizer, to create an appropriate
transformation in the virtual environment. Wexelblat’s generic gestural interface is thus adaptable to particular scenarios.

A more specialized approach to gesture modeling is adopted by Surface Drawing (Schkolne et. al. 2001). The goal of Surface Drawing is to develop a method for creating three-dimensional shapes that mimics traditional artistic processes. The system uses a semi-immersive environment called the Responsive Workbench, at which the user wears stereoscopic glasses and instrumented gloves (Figure 8). Drawing occurs in the same physical space in which the model is constructed. When the artist moves her hand in free space above the workbench, a three-dimensional stroke is created and holographically displayed in the air. The drawing thus created can be manipulated by a variety of physical tools: tongs let the user move and rotate the drawing, an eraser lets the user remove selected portions, and a magnet tool facilitates the smoothing of either a single stroke or an entire drawing.

Figure 8. Surface Drawing: mark-making above the Responsive Workbench. From Schkolne et. al. (2001:Figures 1 and 3).
Surface Drawing is distinct from other surface-modeling applications in the method it uses to create geometry. Rather than manipulating B-splines, users create geometry directly with their hands. The path created by the designer’s hand becomes digital geometry, a method that provides direct control over the creation of three-dimensional form. The spatial coincidence of hand and mark gives a sense of immediacy by uniting physical motion with mark-making. According to the authors, the new medium facilitates conceptual exploration by giving artists a more emotional relationship with their tools and a better physical understanding of their workspace (Schkolne et. al. 2001:261).

Despite their potential in earth-forming applications, there are also drawbacks to gesture-based interfaces. Most systems lack haptic feedback of any kind. This deficiency is a particular disadvantage to systems that seek to extend physical, gestural methods – like those used in sculpting or model-making – into the digital realm. In addition, some gestural interfaces – like GestureVR – separate the gesture and display spaces, which may hinder users from properly orienting themselves. Gesture-based interfaces are also quite divergent in their hardware requirements and implementation strategies. Some – such as GestureVR – adopt machine-vision approaches, while others – such as Surface Drawing and Wexelblat’s system – utilize instrumented clothing and complex gesture-recognition algorithms. Despite these drawbacks, gestural interfaces still have the potential to more closely approximate the methods used in sandboxes than traditional GUIs.
2.4 METHODS FOR INTEGRATING DESIGN AND ANALYSIS TASKS IN DIGITAL COMPUTATIONAL ENVIRONMENTS

In recent years, a number of research projects have attempted to integrate design and analysis tasks in digital computational environments. Some projects have attempted to create novel, visual interfaces to geographic data, while others have combined CAD and GIS components to develop hybridized applications. While many of these systems provide much-needed support for three-dimensional visualization, the majority of them do not allow design and analysis to take place in the same environment. Instead, they require a designer to first make changes in a separate module before visualizing those changes in a separate environment. As a result, design and analysis are not integrated as closely as they could be.

An example of a hybrid GIS-CAD application is the system developed by Mayall et. al. (1994), in which the three-dimensional modeling capabilities of a CAD application were integrated with a GIS database in order to assess the visual impact of proposed landscape changes. Objects in the landscape are modeled as elements in a CAD system using their attributes as stored in the GIS database. These attributes include building heights and colors, and vegetation type and size. The system uses a simple scripting language to create three-dimensional representations of each visible object in the database.

In an example application of the project, Mayall used the system to simulate the impact of a zoning change on urban development on a small island (Figure 9). Changes to the zoning codes were made in the GIS database, and the subsequent visualizations illustrated the potential impacts of the zoning changes on the visual quality of the island.
Figure 9. Mayall’s GIS-CAD hybrid system. CAD files (above) and 3D visualizations (below) used in generating before (left) and after (right) images. From Mayall et. al. (1994:47-48).

Mayall’s system was successful in enabling users to evaluate the visual impact of design decisions by integrating geographic data and three-dimensional visualization; however, there are drawbacks to the system. The GIS and CAD modules are only loosely coupled to one another, and this incomplete integration still results in lag time between design and analysis. Furthermore, all design changes must be made in the GIS database, rather than in the CAD-based visualization environment. This flaw is perhaps an inconvenience for designers who prefer a more visual approach to landscape design.

In a similar project, Ervin (1992) connected the three-dimensional visualization capabilities of CAD with the analytical power of GIS to address a broader range of
landscape design and planning issues. The goal of the project was not to generate photorealistic simulations but to produce a digital equivalent of the designer’s sketch by enabling designers to visualize the impacts of changes proposed in a GIS environment. The hybrid CAD-GIS system was applied in the context of a landscape design studio. After designers constructed vulnerability and impact assessment models using GIS, the software translated the GIS grid files into three-dimensional objects using a library of predefined symbols – buildings, houses, vegetation, and paved surfaces (Figure 10).

Figure 10. Three-dimensional objects generated by Ervin’s symbol library. From Ervin (2001:Figure 4).

These symbols corresponded to unique land use codes in the GIS database and could be draped over the landform to provide a rough visualization of the design. The hybrid system also allowed designers to view the impacts of their proposed changes. A simple
color scheme was used to paint different areas of the landscape according to the type of impact experienced by each. Impacts could then be assessed according to their relationship to elements in the landscape (Figure 11). The system’s CAD component

![Figure 11. Three-dimensional scene from Ervin’s system showing impact values painted on landscape. From Ervin (2001:Figure 3).](image)

enables a designer to illustrate the three-dimensional qualities of a land-use plan, while the GIS component adds the computational power that is often lacking in site-scale design. The disadvantages of this approach are similar to those of Mayall’s project. The components are loosely coupled, and all designing must take place in the GIS database before it can be analyzed three-dimensionally.

Like Ervin’s project, the Virtual LA simulation environment is a hybrid CAD-GIS system (Liggett and Jepson 1995). The goal of the project is to act as an integrated three-dimensional urban simulation system for the city of Los Angeles. In contrast to Ervin’s system, photorealism is at the heart of Virtual LA. To build the system, aerial
photographs were combined with street-level images and three-dimensional geometry to create a real-time virtual model. A locally developed three-dimensional visual simulation engine was integrated with AutoCAD and ARC/INFO GIS. This CAD-GIS link enables a user to dynamically query and display information from the GIS database while moving through the three-dimensional model (Figure 12). The integrated system makes the process of visualizing the city easier and faster than could be achieved by any of the components alone, in turn allowing designers to quickly identify problems and evaluate the impact of proposed changes. One of the biggest disadvantages of the system is that changes can not be made directly in the three-dimensional display. To change the model, the designer must either edit the GIS database or construct a separate digital model with a CAD tool and then add it to the scene. While it holds much promise, the focus of the Virtual LA project is more on real-time rendering and visualization of large urban scenes than on integrating design and analysis tasks in the same computational environment.

Figure 12. Virtual LA project interface components: GIS (left) and CAD (right). From Liggett et. al. (2001:Figure 9).
Other visualization systems do not explicitly hybridize CAD or GIS applications but instead use the basic principles of one or the other as points of departure. For example, SmartForest allows the user to interactively edit a forest environment to promote a better understanding of the relationship between design decisions and their impact on forest dynamics (Uusitalo et al. 1997). The system consists of a computer-projected, three-dimensional forest environment in which each tree symbol in the scene is embedded with attributes about its type, size, and health (Figure 13). The user can interactively experiment with design decisions by harvesting and growing trees or by watching the progress of insect outbreaks. According to its authors, SmartForest produces visualizations that are superior to those created in a pure GIS environment. When changes to the forest composition are made, the underlying database is also changed, and all new growth models take these changes into account. The resulting visualization is defensible by reference to this underlying database, as the data values responsible for generating the three-dimensional scene are readily available to the user and can be checked to ensure accuracy. The visualizations produced by SmartForest are thus more powerful tools in communicating complex ecological concepts.

The Toronto University Urban Modeling System (TUUMS) also seeks to enable designers to better understand the decision-making process in landscape design (Danahy and Wright 1988). The TUUMS researchers argue that although computers typically do not support design as intuitively as traditional tools do, they provide enormous quantitative sophistication that should be harnessed. To take advantage of this computational sophistication, TUUMS was designed to enable the user to generate digital
three-dimensional forms while receiving real-time feedback about the effect of the design on factors such as floor-space index and construction cost. The user begins by importing a parcel file and clicking on a particular parcel to select it. Once the parcel is selected, the designer can generate spatial zoning constraints using drop-down menus. Buildings can be added to the parcel and scaled according to the constraints. At the bottom of the screen is a spreadsheet window that provides numeric information about the changing
design. By providing feedback on the spatial and financial impact of a design, TUUMS enables a user to address both design and analysis tasks in a single work environment.

While the above projects, to varying degrees, attempt to integrate design and analysis by developing novel environments for three-dimensional visualization, they all limit themselves to purely digital environments. This limitation is unfortunate for designers accustomed to working with tangible tools. Recognizing this disparity, some scholars have attempted to provide tangible, rather than digital, environments for visualization. For example, Clark and McKeon (1998) developed a system in which digital data is projected onto a physical terrain model. The model was fabricated out of thin sheets of translucent paper, and then deforestation data derived from a GIS database were projected onto the model from underneath it. By combining the digital data with a physical representation of topography, they hoped to promote a better understanding of environmental issues, as well as a stronger sense of place. Ultimately, however, the system is a simple static display of geographic data with no support for making changes directly to either the physical model or the environmental data.

Similarly, Underkoffler and Ishii (1999) developed the Luminous Planning Table (LPT) in an effort to integrate digital, physical, and visual representations of urban environments (Figure 14). The LPT is a physical workbench situated beneath overhead projectors and cameras. The workbench acts as the modeling surface and contains physical objects that can be manipulated by the user. The cameras capture and calculate the changing positions of objects on the workbench as the designer rearranges them. A computer attached to the workbench calculates various attributes associated with the
objects – like sun, shade, wind, and traffic patterns – and these are projected onto the table. In this way, the LPT simulates the impact of a design on its environment while enhancing the design process by providing active feedback in a tangible, design-oriented environment. Even though its visualization environment differs from those of the hybrid GIS-CAD projects, the LPT provides similar testament to the need to create visual representations of quantitative or spatial data, as well as new interfaces to that data.

Recent work in GIS is similarly concerned with user interaction with spatial data. At present, most methods for interacting with geographic data are often tedious, since many of the fundamental ways of working in GIS have nonspatial origins. In fact, some projects concerned with improving access to geographic information have begun moving away from CAD-based interfaces altogether. Max Egenhofer proposes an interface that allows a user to make sketch-based queries of geographic information, particularly in the early stages of the problem-solving process (Blaser et. al. 2000). For example, if a geologist wanted to know the location of an oil deposit, she could simply sketch a rough
cross-section of a typical oil field. The GIS would then query the database to find geographic locations whose spatial relationships were equivalent to those in the sketch. Recognizing that in most GISs visualization takes place only after the initial problem has been formulated, Egenhofer proposes using sketch GIS to introduce visualization at the earliest stages of problem formulation. By providing a spatially-oriented, sketch-based method for accessing geographic data, the authors explicitly recognize the need to support visual interfaces to quantitative data.

2.5 Conclusions

This review of related work highlights some of the advantages and shortcomings of digital earth-forming tools. In light of their familiarity to designers, traditional analog and CAD tools are in many ways useful for earth-forming; however, recent projects like TSS and Leveller™ attempt to transcend some of the limitations associated with traditional methods by developing systems that use alternative metaphors to support earth-forming. These new tools compliment CAD-based methods by supporting design in the early conceptual stages when a user needs the freedom to rapidly and iteratively explore alternatives scenarios.

The traditional GUIs used by most digital tools are somewhat of a disadvantage for earth-forming applications. They typically do not support the easy, intuitive manipulation of three-dimensional form to which designers are accustomed when working with tangible media like sandboxes. Because designers use hand gestures when sculpting in sandboxes, gesture-based interfaces provide an alternative to GUIs. Gestural
interfaces have the potential to enable a designer to extend familiar and comfortable ways of working into the digital realm.

There have been a number of attempts to integrate design and analysis tasks into the same work environment, but seamless integration has been difficult to achieve. The importance of analytical tasks to the design process has been acknowledged by most researchers, but analysis has often been limited to visual feedback. As a result, some of the projects reviewed above – such as Virtual LA – cast themselves simply as systems for “visualization,” while others – like Ervin’s system – interpret analysis more broadly and include the measurement of ecological patterns and processes as well. All of the projects seem to concentrate most of their effort on developing better environments for visual analysis, without providing explicit support for design. Despite this limitation, their desire to combine visual or aesthetic models of the landscape with quantitative ones speaks to an ultimate need to integrate design and analysis. This desire also suggests that, in the words of Ervin (1992:30), “the idea of multiple views of evolving models is the best underpinning for comprehensive computer-aided design, whether it’s called CAD or GIS (or anything else).” Integrating these multiple models of landscape is more efficient when a designer is able to connect design and analysis tasks in the earth-forming process.
CHAPTER 3
THE DIGITAL Sandbox: A Prototype System for Digital Earth-Forming

3.1 INTRODUCTION

The Digital Sandbox is a gesture-based application that integrates design and analysis tasks in support of digital earth-forming. It enables a designer to sculpt a digital mesh, add trees and buildings to the mesh, reposition the objects as needed, and run a simple stormwater accumulation model to predict flow paths over the surface of the mesh. The designer uses a predetermined number of gestures, which are mapped directly to specific commands, to accomplish these tasks. The system has been designed as a rapid, proof-of-concept prototype to explore the viability of such an application in digital earth-forming contexts. This chapter explains the design and implementation of the system’s main components and provides a design scenario to demonstrate how the tool is used in digital earth-forming.

3.2 SYSTEM OVERVIEW

The system uses a machine-vision approach similar to that developed by Segen and Kumar (1998). It has two primary physical components: a gesture-input space and a model space. The gesture-input space consists of a simple, desktop video camera positioned approximately one meter above a uniformly white tabletop (Figure 15). All gesturing takes place between the camera and the tabletop. The user wears a black glove, and the system uses the contrast between this black glove and the white background to
isolate the user’s gestures. The hand must be oriented parallel to the tabletop when gesturing. The camera reads a continuous data stream and simultaneously communicates the data to the computer. The model space consists of two program windows that are rear-projected onto a translucent screen. One window displays a simple three-dimensional scene, as well as the gestures captured by the camera (Figure 16). The smaller gesture-recognition window enables the user to determine whether the system is accurately recognizing the gesture.

The system is implemented in C++ and consists of four primary software components: Vision SDK 1.2, a low-level library of methods used to acquire live images, developed by Microsoft’s Vision Technology research group (Microsoft Corporation 2001); Microsoft Direct3D, a graphics and rendering package; the Gesture Modeling interface, developed by Ariel Kemp and Mark Gross at the University of Washington’s
An explicit object-oriented approach was adopted in designing and implementing the Digital Sandbox. Object-oriented programming is an approach to software design that divides an application into individual, self-contained conceptual modules called objects (Carrano et. al. 1998:18). These objects contain attributes – like size, color, or shape – and methods that operate on the attributes – like changing the size, color, or shape. Each class performs a well-defined task or set of tasks. Similar types of object are grouped into classes, and these classes are the building blocks of the software.
Object-oriented programming uses three basic principles to simplify the software-writing process. First, encapsulation allows attributes and behaviors of objects to be embedded within classes. As a result, each class is able to function independently of all others. Secondly, inheritance enables a class to inherit properties from other classes to produce a modified, but related, class. Thirdly, polymorphism enables a program to determine exactly how operations will be carried out at run time, based on the particular objects in use at the time and the relationships between them. Together, these three principles enable the programmer to produce modular solutions for programming problems.

In adopting an object-oriented approach for the Digital Sandbox, two general class categories were developed: those used in gesture modeling, and those used to implement the sandbox world. Each category is discussed below.

3.3 GESTURE MODELING IMPLEMENTATION AND CLASSES

3.3.1 GESTURE INPUT AND INITIAL IMAGE PROCESSING

The computer receives input from the camera as a bitmap image. In order to distinguish the glove from the background, the intensity of each pixel in the image is compared to a predetermined threshold value. If the intensity is below this threshold, it is replaced with a maximum value (resulting in a white pixel). Intensities above the threshold are replaced with minimum values (resulting in black pixels). To distinguish the hand from the rest of the image, the system searches for the largest contiguous area of black pixels and draws a bounding box around it.
3.3.2 Gesture-Command Mapping

The system recognizes a total of seven discrete gestures (Figure 17), each of

![Gestures](image)

which is mapped directly to a single command (Table 1). These gestures are used by the designer to perform specific earth-forming tasks. For example, to sculpt a digital landform, the designer uses the “point” gesture to deform the mesh. The system reads the gesture as “point,” calculates the coordinates of the designer’s index finger, and then deforms the mesh by disturbing the points around these coordinates. Using this method,
Table 1. Gesture-command associations used by the Digital Sandbox.

<table>
<thead>
<tr>
<th>Gesture</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fist</td>
<td>Moves a building by following the user’s hand around the screen. Used in combination with Five to reposition buildings in the sandbox.</td>
</tr>
<tr>
<td>Five</td>
<td>Selects and deselects a building. Used in combination with Fist to reposition buildings in the sandbox.</td>
</tr>
<tr>
<td>Gun</td>
<td>Adds a tree to the sandbox.</td>
</tr>
<tr>
<td>Pinch</td>
<td>Selects a tree and repositions it in the sandbox.</td>
</tr>
<tr>
<td>Point</td>
<td>Deforms the digital sand.</td>
</tr>
<tr>
<td>Thumbs up</td>
<td>Initiates the stormwater accumulation model.</td>
</tr>
<tr>
<td>Two</td>
<td>Adds a building to the sandbox.</td>
</tr>
</tbody>
</table>

an input gesture, along with an associated command, produces some action in the digital environment. This gesture-command association is the basis of the system.

3.3.3 Pattern-Matching Using Stored Template Images

Once the system isolates the gesture and circumscribes it with a bounding box, the thresholded image inside the box is compared to a set of stored template images to determine what gesture the designer is using. The templates are a set of bitmap images of the hand gestures that were constructed using input images from the video camera (Figure 18). In PhotoShop, each image was subjected to the same threshold process described above (section 3.3.1), cropped with a similar bounding box, resampled, and subjected to a Gaussian blur. The important spatial features of the image were then colored to make the comparison with input images as simple as possible. White pixels were replaced by red, black pixels by blue, and finger axes were drawn in green. These axes were then outlined in yellow. Residual wrist and lower-arm areas were colored
Two main classes are used in this pattern-matching process: \textit{KnownImage} and \textit{KnownImageSet}.

3.3.3a KnownImage

The KnownImage class functions as the data structure for each stored template. The class is responsible for storing information about the dimensions and spatial structure of each image and uses a data type from the VisionSDK library to do so.

3.3.3b KnownImageSet

The KnownImageSet class organizes the template images and compares them to input images from the camera. The system performs real-time image comparison on a
per-frame basis. As each input image is compared to the templates, a score is generated that determines which template is the appropriate match.

Three major steps are used to compare input images to the templates. First, if the aspect-ratio of the images differs by more than a predetermined amount, the template is deemed an inappropriate match and the comparison stops. Secondly, percentages of red and blue pixels in the template are compared to the percentages of white and black pixels (respectively) in the input image. The intensity values of gray pixels are compared to pixels in the input image as well. If the percentages of each color are lower than a predetermined value, the template is deemed inappropriate. Otherwise, the score is incremented by an amount proportional to the percentage of matching pixels. Finally, the system compares the eight nearest neighbors of each green and yellow pixel in the template to their corresponding pixels in the input image. It calculates the percentage of green pixels with black neighbors and yellow pixels with white neighbors, and the score is then further incremented by an amount proportional to these percentages. The system proceeds stepwise through each stored template to form a ranked list of possible matches.

Once the list of potential template matches has been formed, the weakest matches are discarded. The system then selects the template that corresponds to the highest-ranked gesture of the last five frames and replaces the last gesture in the list with the new one. The most common gesture in the list is then deemed the gesture of the current frame. This five-frame “buffer” lends stability to the system by allowing it to disregard the short-term (< 5 frames) fluctuations caused by inappropriate matches. Finally, the system calculates the spatial location of the hand in the gesture space. The x, y
coordinates are calculated using the corners of the bounding box, and the z-coordinate is estimated by measuring the total number of pixels in the hand.

3.4 **Digital Sandbox Implementation and Classes**

The general design strategy for the Digital Sandbox was to develop a set of classes that would represent a traditional, physical sandbox. The sandbox’s sand is represented by a digital mesh class and its toys are represented by simple geometric objects like cones, cylinders, and cubes. The sandbox itself is represented by a separate class. In addition, each of the objects in the sandbox plays some computational role in the stormwater accumulation model. To make the toys useful in computing the model, they are also embedded with some of the attributes of the real-world objects they imitate. These classes, along with their attributes and methods, are discussed below.

3.4.1 **TheSandbox**

The TheSandbox class is the system’s control structure: it controls the creation of new toys and organizes user interaction with each of them. It contains a digital mesh and the objects that are active in the sandbox at any given time. The key components of the class are as follows:

**Attributes:**
- *Toy* – An array of SandboxToys. Toys that are active and being played with in the sandbox are stored in this array.
- *CurrentlyPlaying* – Boolean variable indicating whether the user is playing with any of the toys in the sandbox.
- *ActiveToy* – A pointer to the toy being played with by the user.

**Methods:**
- *CreateNewToy* – Instantiates an object of type SandboxToy (section 3.4.2). The toy can be either a tree or a building, depending on user input.
- **AddToyToSandbox** – Adds a newly instantiated toy to the array of SandboxToys.
- **InteractWithToys** – Controls and directs user interaction with the toys by carrying out appropriate commands based on the user’s gesture. The commands enable TheSandbox to create new toys, select and reposition toys in the sandbox, deform the mesh, and run the stormwater accumulation model.
- **IntroduceTheNeighbors** – Notifies the mesh of the presence of other toys in the sandbox. This method finds the position of each toy in the sandbox, snaps it to the mesh, and modifies the interception or infiltration of the mesh beneath the toy (see section 3.4.4 for more on interception and infiltration).

### 3.4.2 SandboxToy

The SandboxToy class is the base class from which the Mesh, Tree and Building classes are derived (Figure 19). Its responsibilities are to interact with the sandbox and to keep track of its location while doing so. The key components of the class are:

**Attributes:**
- **ID** – Identifies what type the toy is. The toy can be either a Mesh, Tree, or Building.
• **PlayMode** – A boolean value that keeps track of whether the user is playing with the toy or not.

**Methods:**
• **Interact** – Controls the user’s interaction with the toy based on gestural input. The implementation of this method is deferred to derived classes.

### 3.4.3 Mesh

The Mesh class represents the sandbox’s “digital sand” – it is the surface that is sculpted by the user and, as such, is the only toy that is permanent (unable to be created or destroyed by the user). The digital mesh formed by the class is a Triangulated Regular Network (Ervin 1994) that is subdivided into a series of discrete MeshTriangles (section 3.4.4). These triangles are organized into an array of rows and columns for efficient storage and manipulation. In addition, the Mesh class is embedded with a stormwater accumulation model.

The main tasks of Mesh are to deform itself in response to gestural input and to run the stormwater accumulation model when initiated by the user. To deform itself, Mesh modifies the z-coordinates of its component triangles using the Gaussian equation,

\[
f(\mu) = \left( \frac{1}{(2\pi\sigma)^{1/2}} \right) e^{-\left( \frac{(\mu - \mu)^2}{2\sigma^2} \right)}
\]

where \(\mu\) is the mean of the distribution (the point at which the user touches the mesh) and \(\sigma\) is the variance. The \(\sigma\) value is important because it controls the shape of the deformation by calculating the appropriate number of points that should be disturbed around \(\mu\) (Figure 20). Currently, a value of 0.2 for \(\sigma\) is hard-coded into the system, since this value was found through trial-and-error to give the deformation a moderate bell
Figure 20. Gaussian equation variables used to determine the shape of Mesh deformation.

shape (not excessively pointed or flat); however, this could easily become a variable controlled by the user.

The stormwater accumulation model controlled by Mesh is a simple, modular model that I designed to predict flow paths across the surface of the mesh. Unlike more complicated approaches (e.g., Amer and Blais 2000) that seek to route flow by defining the structural features of the digital terrain (like channels, ridges, and valleys), my approach treats each discretized unit of the mesh as an independent component. Each unit is responsible for receiving water from various sources and then routing flow across its surface.

The approach utilized here is similar to the one used by Silfer et. al. (1987) to route flow over a Triangulated Irregular Network. To route flow across the network of triangles, the model uses simple, local rules derived from the topological relationships between adjacent triangles. Because these triangles are always coplanar and joined at
their edges, boundary conditions are relatively easy to specify. Thus, deriving local rules that describe flow between triangles is much easier for a triangulated model than for a gridded Digital Elevation Model (DEM), whose fundamental unit is a point (Wilson and Gallant 2000). Surfaces created from point matrices are potentially much more complex than those created from planes, since adjacent surfaces may not always be coplanar. Since the individual, triangulated units of the mesh are always coplanar, the model can easily handle all possible boundary conditions for every triangle. Thus, to route flow across the entire mesh, the model only needs to be able to describe flow across an individual triangle.

Mesh uses a multi-step process for computing the stormwater accumulation model. First, the mesh triangles are sorted based on their elevation. Then, beginning with the triangle with the highest elevation, the model allows the triangle to compute its own runoff (see section 3.4.4 for more on runoff calculation). Next, the model routes the runoff from the triangle to its nearest downhill neighbor. To determine this neighbor, the model calculates the orientation of the line of steepest descent by projecting the triangle’s aspect onto the x-y plane. Based on the orientation of this line, flow is routed in one of six possible directions – either one of the triangle’s three faces or one of its three corners (Figure 21). The model then determines whether this neighbor is downhill of the triangle. If the neighbor’s elevation is lower than the triangle’s, flow is routed; if not, the model searches a wider neighborhood for the next lowest triangle lying along the line of steepest descent. If a lower neighbor is found, flow is routed to it; otherwise, the triangle is deemed a sink and the flow is absorbed by it. If neighboring triangles are facing one
Figure 21. The six possible directions in which a triangle can route flow.

another, the model averages their aspects and uses this line to continue routing flow. The
model proceeds to the next triangle in the sorted list and the procedure is repeated. Using
these simple, local rules, flow is routed across the entire surface of the mesh from the
highest elevation to the lowest.

The key components of the class are:

**Attributes:**
- **Triangle** – The data structure that organizes the Mesh into a double array of
  MeshTriangles (section 3.4.4).
- **x_extent** – Number of columns in the mesh.
- **y_extent** – Number of rows in the mesh.

**Methods:**
- **Interact** – Controls user interaction with the mesh. There are two possible
  scenarios. If the user is pointing at the mesh, it is deformed appropriately. If
  the user gestures with “gun,” it initiates the stormwater accumulation model.
- **ModifyMeshGauss** – Applies a Gaussian disturbance to the mesh by
determining which triangles should be modified.
- **RunStormwaterModel** – Initiates the stormwater accumulation model and
  controls it.
• **QuickSortTriangles** – QuickSorts (Carrano et. al. 1998:417) all MeshTriangles from highest to lowest elevation.

• **RouteRunoff** – Routes the runoff of each MeshTriangle using the process described above.

• **EvaluateFacingNeighbors** – Determines the flow path if two triangles are facing one another. Routes flow to the next downhill neighbor that lies along this flow path.

• **FindNextLowestNeighbor** – Searches a wider neighborhood to find the triangle with the lowest elevation along the flow path. If none is found, the triangle absorbs the flow instead of routing it.

### 3.4.4 MeshTriangle

The MeshTriangle class makes up the discrete subdivisions of Mesh. Each triangle is responsible for several related tasks: modifying its vertices when the mesh is deformed by the user; calculating its gradient, aspect, and elevation; computing its stormwater runoff; and setting its color and fill mode based on its runoff value.

Each triangle computes its stormwater runoff according to the equation (Burrough and McDonnell 1998:206),

\[
\text{Runoff} = \text{Input} - \text{Interception} - \text{Infiltration} - \text{Evaporation} + \text{Runon} \tag{2}
\]

where Runon is the quantity of runoff routed to the triangle by its uphill neighbor. Currently, the numeric values used by the model are arbitrary and hard-coded into the system; however, the user could easily control them in a dialogue box.

Each triangle uses its computed runoff value to determine its fill mode and color. The triangle is painted solid if the runoff exceeds a threshold value. Conversely, the triangle remains in wireframe mode if this value is not exceed. The threshold value is currently hard-coded into the system but could also be specified by the user. A graded,
choroplethic color scheme is used to paint the triangle. Less saturated colors correspond to lower runoff values, while higher-saturation colors correspond to higher values.

The most important components of the class are as follows:

Attributes:
- **Elevation** – The height of the triangle above the zero-elevation baseline established by the coordinate system of the digital scene.
- **Aspect** – Orientation of the triangle’s line of steepest descent.
- **Gradient** – Rate of change of elevation from the highest vertex of the triangle to the lowest.
- **InputPrecipitation** – Quantity of precipitation falling on all triangles in the mesh. The value is currently hard-coded into the stormwater model.
- **Evaporation** – Quantity of water evaporating above the triangle. This variable is not used by the current implementation of the stormwater model. A more environmentally sophisticated model that included ways to specify or create atmospheric or other ecological variables could make use of this variable.
- **Interception** – Quantity of precipitation intercepted by objects located directly above the triangle.
- **Infiltration** – Quantity of precipitation infiltrating the triangle.
- **RunoffFromNeighbors** – Quantity of runoff the triangle receives from neighbors.
- **RunoffToNeighbors** – Quantity of runoff the triangle routes to a neighbor.

Methods:
- **CalculateElevation** – Computes the elevation of the triangle by averaging the z-coordinate values of its vertices.
- **CalculateAspect** – Computes the aspect by measuring the angle of the triangle’s normal projected onto the x,y plane.
- **CalculateGradient** – Computes the gradient by finding the change in elevation of the triangle’s normal in the x,y plane.
- **ModifyVertices** – Computes new normals for the triangle’s vertices when its z-coordinates are modified by mesh.
- **SetInterception** – Updates the interception of the triangle if a tree is located on top of it.
- **SetInfiltration** – Updates the infiltration of the triangle if a building is located on top of it.
- **ComputeRunoff** – Calculates **RunoffToNeighbors** using equation (2).
- **SetRunoffFromNeighbors** – Updates **RunoffFromNeighbors** if flow is routed to the triangle from an uphill neighbor.
- **UpdateColor** – Sets the color of the triangle based on its runoff values. Uses a graded, choroplethic scheme to do so.
- **SetFillMode** – Sets the fill mode of the triangle to either solid or wireframe.
3.4.5 **Tree**

The Tree class is derived from SandboxToy and contains a TrunkType (section 3.4.6) and a CanopyType (section 3.4.7). It is responsible for setting the attributes of its trunk and canopy and for intercepting rainfall based on these attributes. Although the class currently allows the user to add only one type of tree to the sandbox, it is intended to be an extensible framework for creating multiple tree types. By encapsulating the attributes of its physical counterpart, each tree type would interact differently with the mesh (e.g., by intercepting different quantities of rainfall). The main components are:

**Attributes:**
- *Trunk* – A TrunkType that represents the tree’s trunk.
- *Canopy* – A CanopyType that represents the tree’s canopy.

**Methods:**
- *Interact* – Moves the tree using the coordinates of the user’s hand position.
- *SnapToGrid* – Automatically snaps the tree to the mesh when released by the user.

3.4.6 **TrunkType**

The TrunkType class constitutes half of the Tree class. Its main responsibilities are to maintain its dimensions and to move itself when the entire tree is moved. It contains simple methods to accomplish these tasks. As a component of the Tree class, TrunkType is also an extensible framework for encapsulating the behaviors and attributes of various trunk types.

3.4.7 **CanopyType**
The CanopyType class constitutes the other half of the Tree class. Like TrunkType, its main tasks are to maintain its dimensions and to move itself when the tree is repositioned by the user. CanopyType is intended to be an extensible framework for a variety of canopy types. In this framework, each canopy type would be responsible for determining how it intercepts rainfall.

3.4.8 BUILDING

The Building class is derived from SandboxToy. The main task of the class is to decrease the infiltration values of mesh triangles downhill of it, since buildings are typically impermeable to rainfall. It is responsible for maintaining its dimensions and location in the sandbox. It uses simple methods to accomplish these tasks; however its Interact method is more complex than Tree’s method. Because repositioning a building involves two gestures – “five” and “fist” – the Interact method must keep track of a sequence of recent gestures to determine whether the user is attempting to select the building, release the building, or is in the process of moving the building.

3.5 APPLICATIONS OF THE DIGITAL SANDBOX

Designing with the Digital Sandbox is a straightforward and direct process. In contrast to the scenario described in Chapter 1 (section 1.2), the designer uses simple freehand gestures to create a digital landform. Moreover, the separation between design and analysis tasks is minimized – the user is able to design a landform and then run a stormwater accumulation model on the landform within the same work environment.
A typical design session might begin with the user donning the glove and gesturing in the input space. The camera captures the gestures and displays them on the monitor inside of the model space. To sculpt an earth form, the designer simply deforms the mesh by pointing at it (Figure 22). She uses a single, simple gesture to sculpt the mesh – no contour lines are needed. The earth-forming continues until the designer creates a suitable landform.

At any point in the design process, the user can analyze how the design will impact the flow of stormwater on the site. She simply makes a “thumb” gesture to initiate the stormwater accumulation model (Figure 23). The system evaluates the elevations and orientations of each triangle in the mesh and routes flow across its surface.
Figure 23. Initiating the stormwater accumulation model with the “thumbs up” gesture. The mesh is shown before the model is run (left) and after (right), with stormwater values painted directly on the mesh.

To indicate the location of flow paths, the system colors all those triangles whose runoff exceeds a predetermined threshold value. The Digital Sandbox renders these flow paths directly onto the landform by painting each triangle according to its accumulated value. This method enables the designer to visualize the location and relative concentration of flow. This information, in turn, enables her to make better decisions about the impact of the design.

If the designer decides the stormwater needs to be redirected, she can simply alter the flow paths by continuing to sculpt the mesh. She can iteratively design the landform and analyze flow paths across it (Figure 24). Alternatively, if the designer is satisfied with the flow paths but still concerned about the concentration of water on various parts
of the site, she can add trees to the landform to intercept some of the rainfall. She could use the “gun” gesture to add a single tree to the sandbox, and then reposition it with the “pinch” gesture (Figure 25). The process can be repeated to create additional trees. Once the trees are positioned on the site, the designer can run the stormwater model again. By intercepting rainfall, each tree decreases stormwater accumulation immediately downhill of itself. When clustered in a particular area of the site, the trees significantly decrease surface flow in that area (Figure 26). Again, the user is able to work iteratively between design and analysis within the same tool.

The designer uses a similar process to add buildings to the sandbox. The “two” gesture creates a simple building (Figure 27). To move the building, the designer utilizes a combination of gestures: “five” and “fist” (Figure 28). The “five” gesture selects the building when the designer brings her hand close to it. Then, she quickly forms a “fist”
Figure 25. Creating a tree and moving it onto the mesh (clockwise from upper left). The designer uses the “gun” gesture to create a tree (upper left) and the “pinch” gesture to grab and move the tree (upper right). When the tree is positioned on the mesh (lower right), the designer releases it (lower left).

to grab the building. Once she grabs the building she can reposition it on the landform. When she is ready to release it, she makes another “five” gesture and the building is
Figure 26. A group of trees altering stormwater accumulation, before (left) and after (right) the model is run. Note the disappearance of red triangles immediately downhill of the trees (right).

Figure 27. Creating a building with the “two” gesture.
Figure 28. Placing the building on the mesh using a combination of the “fist” and “five” gestures (clockwise from upper left). The designer selects the building with the “five” gesture (upper left) and grabs it with the “fist” (upper right). When the building is in place, the “five” gesture releases it (lower right) in its new position on the mesh (lower left).

deselected. Now when the designer runs the stormwater model she can see the effect of
the building on runoff values downhill (Figure 29). Because it is impermeable to rainfall,

![Figure 29. A building altering stormwater accumulation on the mesh, before (left) and after (right) the model is run. Note the appearance of red values immediately downhill of the building (right).](image)

the building increases stormwater accumulation in the area immediately downslope of it.

As the above scenario illustrates, the Digital Sandbox supports digital earth-forming by integrating design and analysis tasks into a single tool and by enabling the designer to use freehand gestures in space. The designer is able to sculpt a digital mesh, create objects and add them to the sandbox, and run a stormwater accumulation model. The iterative processes of design and analysis are more closely coupled.

### 3.6 Conclusions

The Digital Sandbox is a proof-of-concept prototype tool that supports digital earth-forming. Its gesture-based interface enables a designer to use hand gestures in a
digital modeling environment. It utilizes a machine-vision approach to implement the interface. A video camera captures input, and the system then determines gesture type through a pattern-matching process. Each gesture is directly mapped to a specific command and is used to carry out a particular action in the digital model. In addition, the Digital Sandbox explicitly attempts to bridge the gap between design and analysis tasks by enabling a user to design a landform and then analyze stormwater flow over that landform. The system adopts an object-oriented approach to develop a set of C++ classes embedded with the behaviors and attributes found in the real-world counterparts of tangible sandbox toys. Because the stormwater model is embedded within the mesh, the user can run the model at any time in the design process. This integration brings design and analysis tasks closer together in a single earth-forming environment.
CHAPTER 4
CONCLUSIONS AND IMPLICATIONS

4.1 SUMMARY

The goal of this study is to design and build a digital computational tool that provides better support for earth-forming. It is motivated by the observation that current earth-forming tools, though useful to landscape architects in many ways, limit the creativity and flexibility of the earth-forming process. I suggest that it is difficult for designers to integrate the aesthetic and ecological objectives of an earth-forming project because no tool enables them to perform both design and analysis tasks in the same environment. In response to these concerns, I sought to develop the Digital Sandbox to provide better support for earth-forming by integrating both design and analysis into a single work environment. The resulting system enables the user to design a digital landform and simultaneously analyze the flow of stormwater over that landform using freehand gestures in space.

Conceptually, the Digital Sandbox is situated at the juncture of three primary issues in design computing: methods for digital earth-forming, gesture-based methods for digital modeling, and methods for integrating design and analysis tasks in digital computational environments. A review of work in these areas revealed three primary conclusions. First, although relatively few innovations have occurred in the area of digital earth-forming, some recently developed tools are definite improvements over existing CAD-based methods, particularly for design in the early conceptual stages.
Secondly, traditional GUIs are familiar to most designers and have been the basis for almost all advances in software design in recent decades; however, because they require designers to relinquish the gestural methods used when working in sandboxes, traditional GUIs are somewhat of a disadvantage for digital earth-forming. Gesture-based interfaces have potential for introducing these methods into digital earth-forming environments. Lastly, a number of projects have attempted to integrate design and analysis, especially under the rubric of “visualization.” Seamless integration has been difficult to achieve, and as a result, design and analysis tasks remain separated or at best only loosely coupled.

The metaphor of a physical sandbox serves as the conceptual framework for the design of the Digital Sandbox. The system adopts an explicit object-oriented approach to develop a set of classes that represent the objects used in a traditional, physical sandbox. The classes are embedded with the behavior and attributes of their real-world counterparts to make them useful in computing a stormwater accumulation model. The user manipulates these objects using gestures in three-dimensional space. Each gesture is directly mapped to a particular command, and this association enables the user to carry out actions in the digital sandbox. In this way, the user can sculpt a digital mesh, add trees and buildings to the sandbox, and run the stormwater accumulation model.

4.2 CONCLUSIONS

The Digital Sandbox was designed and implemented with two main goals in mind: to more closely integrate design and analysis tasks in a single computational environment; and to utilize a gestural interface to approximate the methods used in a
traditional, physical sandbox. The system meets these goals in several ways. It creates a seamless junction between design and analysis by enabling the user to design a landform and run a stormwater model on the landform. The results of the analysis are painted directly on the model, allowing the designer to use the information to make a more intelligent decision about the impact of the design. In addition, the gesture-modeling interface uses simple, easy-to-learn gestures to enable a user to directly and easily create three-dimensional digital landforms. The interface reduces the lag time between design conception and execution by letting the user design directly in a three-dimensional representation of the landform. Informal evaluation of the tool with various user groups (designers, non-designers, adults, teenagers) suggests that most people are able to immediately begin working with the tool with little training or prior experience.

The Digital Sandbox also has a number of shortcomings. Various compromises were made to enable the gesture-modeling interface to be quickly implemented, and these compromises create a number of restrictions on the user’s ability to use gesture naturally. The video camera has a relatively small field of view, which physically constrains the dimensions of the gesture-input area. As a result, it is often difficult (sometimes impossible) to access the edges of the digital model to deform the mesh and reposition objects. Furthermore, the user’s hand must be kept oriented perpendicular to the camera at all times, and no attempt is made to account for the roll, pitch, or yaw of the hand. This constraint reduces the degrees of freedom available to the user. In addition, the user is limited to using only those gestures for which there are stored templates and is unable to create new gestures. Wearing the black glove is also a minor inconvenience.
Additional compromises were made in developing the system as a rapid, proof-of-concept experiment. The user has limited control over the shape and extent to which the mesh is deformed. Consequently, the mesh-sculpting process is neither as sophisticated nor as flexible as it could be. The system also lacks the ability to import outside data sources, such as Digital Elevation Models. The mesh provided by the system lacks some of the real-world landscape attributes with which designers typically work, like preexisting topography. Additionally, the stormwater model is currently hard-coded into the system, and the user has no control over input parameters or other variables. Because the details of its parameters, variables, and assumptions are hidden from the user’s view, it might be difficult for landscape architects to accept the results of the model. And while the model is based on sound principles, it is neither as scientifically accurate, nor as environmentally comprehensive, as stormwater models developed by geographers and hydrologists.

A complaint likely to be raised about the system is that the physical scale at which it models stormwater runoff is unrealistically small. Designers are accustomed to measuring stormwater runoff, as well as other ecological processes, at much larger scales using aggregated patches of vegetation rather than individual trees. The scales at which these processes are modeled is usually determined by the resolution of remotely-sensed or digital GIS data. In this light, the simulation employed by the Digital Sandbox does not conform to standard practice. Instead, I defy this practice in order to make an important point: ecological processes should be computable at any physical scale, but especially the small scales at which designers typically design. Such simulations can
provide site designers with useful information that would be difficult to come by otherwise.

There are a number of ways in which the Digital Sandbox could be improved. To enable the designer to reach the edges of the digital model, either the physical dimensions of the gesture space could be increased by adding additional cameras to the system or “pan” and “zoom” features could be implemented. Additional cameras could also be used to capture the user’s hand in profile, enabling the system to calculate roll, pitch, and yaw. This information would, in turn, provide the user with greater degrees of freedom in manipulating the digital model and would be a closer approximation of the methods used in physical sandboxes. The system would have further flexibility if the gesture modeling interface were designed more generically. For example, a user could train the system to recognize gestures beyond those specified by the system, in effect creating additional templates that would be perfectly suited to the user’s own gestural predilections. In addition, it would be useful to develop more complex image-processing algorithms capable of detecting the user’s hand without the user needing to wear a glove. Although designing the algorithms might be somewhat time-consuming, it would strengthen the case for using machine-vision based approaches to gesture modeling, rather than instrumented ones.

Further improvements could refine the mesh-sculpting process by giving the user more control over how the mesh is deformed. These options could specify the shape of the deformation, the spatial extent or width of the deformation, and could allow the user to undo recent actions. In addition, file-reading routines could be written to import
common data sources. This feature would allow a designer to use the tool on an actual site with existing topography. Moreover, if the system used real-world data, the sandbox objects could be embedded with constraints like cut and fill limits or environmentally sensitive areas. Such a constraint-based approach would further integrate design and analysis tasks by enabling the user to explore the constraints imposed not just by stormwater runoff but by other variables as well. Finally, the stormwater model could be made more transparent by allowing the user to control its parameters and variables. Alternatively, the system could enable the user to specify a particular stormwater model to be used, or else it could link to a spatial data engine (like a GIS). These parameters could then be used to determine the behavior and attributes of the sandbox toys.

4.3 IMPLICATIONS

This study has some interesting implications for the ways in which landscape architects design and the tools they use to do so. One of the biggest advantages of traditional analog tools and methods – like drawing, painting, or clay sculpting – is that they engage the human sense of touch in an immediate and expressive way. It seems that many landscape designers are hesitant to leave these physical, tactile methods behind in favor of digital tools, for fear that they will further remove themselves from the landscapes they seek to engage. In many ways, digital design tools have missed this important point, by reducing a designer’s interaction to keystrokes and mouseclicks. And while these interactions serve their purpose, they compel designers to conform their design processes and methods to match those of particular tools.
Gestural interfaces have considerable potential for helping designers interact with digital landscapes in a more direct, physical way. In fact, systems that engage the designer’s hands, arms, and tactile senses are seen by Ervin and Hasbrouk as the harbinger of a revolution in design computing, in which “landscape architects step out of graphics-only application into virtual-reality environments in which users can immerse themselves” (1999:56). The Digital Sandbox, while not immersive, does seek to provide greater engagement of a designer’s body. On the other hand, the limitations of the system make it clear that gesture-based interfaces have a long way to go before becoming a replacement for – or even a widely available alternative to – traditional GUIs. Nonetheless, new interfaces can potentially persuade hesitant designers to transfer their intuitive, physical ways of working into the digital realm.

Why design in digital environments in the first place? Their interfaces are often clumsy and the results many times frustrating. The great advantage of digital environments is that they enable designers to take full advantage of the power of computing for simulating natural phenomena, for computing ecological processes, for providing analysis and feedback in a myriad of ways – tasks that are nearly impossible for humans to do accurately and efficiently. Digital computational tools enable landscape architects to design better landscapes by simulating, testing, and modeling them in a virtual world before they are constructed in the physical world.

This study is also important to landscape architecture because it cuts to the heart of an issue of central importance: the relationship between aesthetics and ecology in the designed landscape. In conceptualizing landscape architecture, and its role and
responsibility in place-making, academics and practitioners alike have tended to polarize themselves into competing camps based upon knowledge, worldview, or predilection: those who see themselves as artists, for whom landscape architecture is about the artful design of space, and those who see themselves more as scientists, the overriding concern for whom is maintaining natural processes and ecosystem health. In the words of James Corner (1992:163):

Both “scientists” and “artists” are, of course, equally correct in their convictions, but quite different. One clearly sides more with nature, and the other with culture. Sadly, however, seldom is there a genuine reciprocity between the two; rarely is there a significant degree of mutual and interactive correspondence between the instrumental and symbolic environments and the activities of human life.

How to integrate these competing perspectives is, according to Laurie Olin, “the central question in landscape architecture” (1992:171).

How might landscape architecture integrate these perspectives in order to more holistically engage and design the landscape? Perhaps one path to integration lies in developing new tools. Not only might new digital computational tools enable designers to more deeply explore the landscape’s natural processes, but in so doing, they may allow designers to express the aesthetics of ecological processes as well. Herein lies the ultimate benefit of design computing – its ability to provide insight into important intellectual issues in landscape architecture. By attempting to support both “artist” and “scientist” in the same environment, the Digital Sandbox provides a heuristic framework for engaging one of the most important of these issues.
BIBLIOGRAPHY


