Anna Petrasova, Brendan Harmon, Vaclav Petras, Helena Mitasova

Tangible Modeling with Open Source GIS

January 26, 2016

Springer

Contents

1 Introduction . . . 1
1.1 The evolution of tangible user interfaces . . . 2
1.1.1 The principles of tangible user interfaces . . . 2
1.1.2 Dynamic shape displays . . . 3
1.1.3 Continuous shape displays . . . 6
1.2 Tangible Landscape . . . 9
1.2.1 Modes of interaction . . . 12
1.2.2 Developing Tangible Landscape . . . 13
1.3 The organization of this book . . . 14
References . . . 14

2 System configuration . . . 17
2.1 Hardware . . . 17
2.1.1 3D scanner . . . 17
2.1.2 Projector . . . 19
2.1.3 Computer requirements . . . 20
2.1.4 Physical setup . . . 22
2.2 Software . . . 24
2.2.1 GRASS GIS . . . 25
2.2.2 GRASS GIS Python API . . . 26
2.2.3 Scanning component . . . 27
2.2.4 Tangible Landscape plugin for GRASS GIS . . . 27
References . . . 31

3 Building physical 3D models . . . 33
3.1 Handmade models . . . 34
3.2 Digitally fabricated models . . . 35
3.2.1 Digital models . . . 36
3.2.2 Laser cutting . . . 38
3.2.3 CNC routing . . . 40
3.2.4 3D printing . . . 42

3.3 Molding and casting 44
3.4 Workflows 46
3.4.1 Selecting a 3D model scale 47
3.4.2 Sculpting a malleable model from data 49
3.4.3 CNC routing a topographic model from contour data . 50
3.4.4 CNC routing topographic and surface models from lidar data 50
3.4.5 3D printing topographic and surface models from lidar data 52
3.4.6 Casting a malleable topographic model with a CNC routed mold derived from lidar data 53
References 54

4 Basic landscape analysis 57
4.1 Processing and analyzing the scanned DEM 57
4.1.1 Creating DEM from point cloud 57
4.1.2 Interpolation with the RST function 58
4.1.3 Analyzing the DEM 59
4.2 Case study: topographic analysis of graded landscape 62
4.2.1 Site description and model 63
4.2.2 Basic workflow with DEM differencing 63
4.2.3 The impact of model changes on topographic parameters 65
4.2.4 Changing landforms 66
References 68

5 Surface water flow and soil erosion modeling 69
5.1 Foundations in flow modeling 69
5.1.1 Overland flow 69
5.1.2 Dam break modeling 71
5.1.3 Sediment transport, erosion, and deposition 72
5.2 Case study: the hydrological impact of development 73
5.2.1 The impact of development on surface water flow 73
5.2.2 The impact of grading on erosion and deposition 75
5.3 Case study: dam breach 76
5.3.1 Site description and input data processing 76
5.3.2 The impact of the road on flooding 78
References 80

6 Viewshed analysis 81
6.1 Line of sight analysis 81
6.2 Case study: viewsheds around Lake Raleigh 82
6.2.1 Site description and model 82
6.2.2 Identifying view points 83
6.2.3 Visibility analysis on DSM using markers 84

6.2.4 Modeling viewsheds from a new building 84
References 86

7 Trail planning 87
7.1 Trail design methodology 87
7.1.1 Least cost path analysis 88
7.1.2 Network analysis 89
7.1.3 Trail slope extraction 89
7.2 Case study: designing a recreational trail 90
7.2.1 Input data processing 90
7.2.2 Computing the trail using the least cost path 92
7.2.3 Finding the optimal trail 94
7.2.4 Mapping trail slopes 95
7.2.5 Alternative trail scenarios 96
References 98

8 Solar radiation dynamics 99
8.1 Solar radiation modeling 99
8.2 Case study: solar irradiation in urban environment 101
8.2.1 The impact of building configuration on cast shadows . 102
8.2.2 The impact of building configuration on direct solar irradiation 102
References 105

9 Wildfire spread simulation 107
9.1 Fire spread modeling methods 107
9.1.1 Input data 107
9.1.2 Fire spread algorithm 108
9.2 Case study: controlling fire with firebreaks 109
9.2.1 Data preparation 110
9.2.2 Scenario with multiple firebreaks 111
References 115

10 Coastal modeling 117
10.1 Modeling potential inundation 117
10.2 Case study: simulating flood risks for barrier island communities 117
10.3 Case study: designing resilient coastal architecture 119
References 122

A Appendix 123
A.1 Applications of Tangible Landscape 123
A.2 Data sources 131
A.3 Starting with GRASS GIS 132

Acronyms

2D	Two-dimensional
2.5D	Two-and-a-half-dimensional
3D	Three-dimensional
API	Application programming interface
CAD	Computer aided design
CAM	Computer aided manufacturing
CLI	Command line interface
CNC	Computer numerical control
DEM	Digital elevation model
DSM	Digital surface model
GIS	Geographic information system
GNU GPL	GNU General Public License
GRASS	Geographic Resources Analysis Support System
GUI	Graphical user interface
HCI	Human-computer interaction
MDF	Medium density fiberboard
NCSU	North Carolina State University
NURBS	Non-uniform rational b-spline
RST	Regularized spline with tension
USLE	Universal Soil Loss Equation
SDK	Software development kit
SED	Simplified erosion and deposition
TanGeoMS	The Tangible Geospatial Modeling System
TIN	Triangulated irregular network
TSP	Traveling salesman problem
TUI	Tangible user interface
UNC	University of North Carolina
USGS	United States Geological Survey
USDA	United States Department of Agriculture
UI	User interface

Chapter 1
Introduction

Fig. 1.1: Tangible Landscape: simulated water flow projected over a physical model of a landscape

The complex, 3D form of the landscape – the morphology of the terrain, the structure of vegetation, and built form – is shaped by processes like anthropogenic development, erosion by wind and water, gravitational forces, fire, solar irradiation, or the spread of disease. In the spatial sciences geographic information systems (GIS) are used to computationally model, simulate, and analyze these processes and their impact on the landscape. Similarly in the design professions GIS and computer aided design (CAD) programs are used to help study, re-envision, and reshape the built environment. However, be-

cause of the unintuitive nature of understanding and manipulating 3D forms in abstract, digital space via a graphical user interface (GUI), technologies like GIS are often so challenging to learn and use that they restrict creativity. The limitations of the technology constrain how we think. Being able to interact more naturally with digital space enhances our spatial thinking, encouraging creativity, analytical exploration, and learning. This is critical for designers as they need to intuitively understand and manipulate information in iterative, experimental processes of creation. With tangible user interfaces (TUIs) like Tangible Landscape one can work intuitively by hand with all the benefits of computer modeling and analytics. This chapter discusses the evolution of TUIs and the development of Tangible Landscape, exploring its motivations, applications, and future directions. This chapter also describes the organization of this book.

1.1 The evolution of tangible user interfaces

Inspired by prototypes like Durrell Bishop's Marble Answering Machine (Poynor, 1995) and concepts like Fitzmaurice et al.'s Graspable User Interface (1995) Ishii and Ullmer (1997) proposed a new paradigm of human-computer interaction – the TUI. They envisioned that TUIs could make computing more natural and intuitive by coupling digital bits with physical objects as Tangible Bits. In their vision Tangible Bits bridge the physical and digital, affording more manual dexterity and kinaesthetic intelligence and situating computing in physical space and social context (Ishii and Ullmer, 1997; Dourish, 2001) Recently, the development of TUIs has gained momentum thanks to new developments in 3D technologies such as 3D scanning and 3D printing.

1.1.1 The principles of tangible user interfaces

We can easily, intuitively understand and manipulate space physically, but our understanding is largely qualitative. We can also precisely and quantitatively model and analyze space computationally, but this tends to be less intuitive and requires more experience. Intuition allows us to perceive, think, and act in rapid succession; it allows us to creatively brainstorm and express new ideas. TUIs like Tangible Landscape aim to make the use of computers more intuitive combining the advantages of physicality and computation.

Physical models are used to represent landscapes intuitively. With a physical model we can not only see its volume and depth just as we would perceive space in a real landscape, but also feel it by running our hands over the modeled terrain. We can shape physical models intuitively – for example we can sculpt landforms by hand, place models of buildings, or draw directly on the terrain. With a physical model, however, we are constrained to a single scale, simple measurements, and largely qualitative impressions.

Digital models are used to visualize, quantitatively analyze, and precisely design landscapes. With digital models we can dynamically visualize different scales and layers of information, study the landscape by computing geospatial analyses and simulations, and precisely edit digital models. Digital models, however, can be unintuitive. A high level of abstraction separates us from a digital model – to interact with this digital abstraction of physical space with a GUI we must translate our intent through our hands to the computer via input devices like a keyboard and mouse and then visualize the results graphically on a display. The digital model itself is a high level abstraction of physical space represented by arrays of values or by mathematical functions. We have to think abstractly and make great conceptual leaps to work through so many layers of abstraction. Understanding and working directly with a digital model is a hard learned skill, a skill that constrains our thinking and transforms creativity.

Tangible user interfaces aim to make interaction between humans and computers more natural and intuitive by giving digital data a physical form and physical form a digital dimension (Fig. 1.1). By coupling the physical and the digital through a feedback loop, we can interact with our physical data directly and very naturally with our bodies – drawing on our highly developed kinaesthetic intelligence.

A landscape modeled as a physical-digital hybrid through a TUI such as Tangible Landscape has the potential to be intuitively modeled, creatively explored, dynamically computed, richly layered with data, and rigorously analyzed. It can have all of the advantages of physical landscape models, geospatial models, and CAD combined. With Tangible Landscape we can shape a hybrid model naturally with our hands to creatively explore ideas as if sketching in multiple dimensions. With a physical model we are constrained to modeling in three-dimensions, to modeling for example topographic form. With a hybrid landscape model we are not limited to modeling topographic form – we can dynamically explore and compute many dimensions of geospatial data in rapid succession with the same model (Fig. 1.2).

1.1.2 Dynamic shape displays

Dynamic shape displays or shape changing displays – another type of TUI – are computer controlled, interactive, physically responsive surfaces. As we interact with the physical surface it changes the digital model and, conversely, as we interact with the digital model the physical surface changes (Ishii, 2008a; Poupyrev et al., 2007). Dynamic shape displays tend to be arrays of pistons and actuated pins that form kinetic, 2.5D surfaces (Petrie, 2006) although there is experimental research into continuous, moving surfaces made of shape changing materials driven by heat, magnetic, or electrical stimuli (Coelho and Zigelbaum, 2010).

Fig. 1.2: Physical, digital, and hybrid physical-digital modeling processes

Aegis Hyposurface The Aegis Hyposurface, an early example of a dynamic shape display, is a generative art installation that uses pneumatic actuators to move a triangulated mesh surface according to an algorithm. It can be either preprogrammed or interactive, moving in response to sensed sound, light, or movement. As it was designed and built at an architectural scale the Aegis Hyposurface has a very coarse resolution for an actuated shape display (Goulthorpe, 2000).

FEELEX The resolution of actuated shape displays is constrained by the size and arrangement of the piston motors and piston rods or pins that move the surface. Project FEELEX, another early dynamic shape display, used linear actuators to deform a rubber plate. The size of the motors – 4 cm – meant that the resolution of the shape display was very coarse. Since the motors are larger than the pins, FEELEX 2 used a piston-crank mechanism to achieve a relatively high 8 mm resolution by clustering the pins while offsetting the motors below. A rubber sheet was stretched over the array of pins to create a 2.5D display for projection. When a user touched the surface they would depress the pins and the pressure of their touch would be recorded as a user interaction (Iwata et al., 2001).

Dynamic Sand Table & Terrain Table The XenoVision Mark III Dynamic Sand Table, developed in 2004, and the Northrop Grumman Terrain Table, developed in 2006, were actuated shape displays that represented topography in 2.5D. In the Terrain Table thousands of pins driven by a motor shaped a silicone surface held taut by suction from a vacuum below into a terrain. The Terrain Table recorded touches as user interactions such as panning

and zooming. As users panned, zoomed, or loaded new geographic data, the actuated surface would automatically reshape within seconds (Petrie, 2006).

Relief Relief is a relatively low-cost, scalable, 2.5D actuated shape display based on open source hardware and software. Given the complexity and thus the cost, maintenance, and unadaptability of earlier dynamic shape displays like FEELEX and the Northrop Grumman Terrain Table, Leithinger and Ishii (2010) aimed to design a simpler, faster system that was easier to build, adapt, scale, and maintain. In the first prototype of Relief an array of 120 actuated pins driven by electric slide potentiometers stretch a Lycra sheet into a shape display. Users can reshape the shape display by pressing or pulling on the actuated pins. The actuators are controlled with open source Arduino boards and a program written in the open source language Processing controls, senses, and tracks all of the pins and their relation to the digital model (Leithinger et al., 2009; Leithinger and Ishii, 2010). The transparency and freedom of open source solutions should make it relatively easy to reconfigure and adapt this system.

Recompose While Relief was initially designed for a simple, highly intuitive interaction – direct physical manipulation (Leithinger and Ishii, 2010) – its next iteration, Recompose, added gesture recognition (Leithinger et al., 2011; Blackshaw et al., 2011). While with Relief users can only directly sculpt the shape display with their hands, with Recompose they can also use gestures to select, translate, rotate, and scale regions of the display. The size and coarse resolution of the actuated shape display mean that only small datasets or subsets of larger datasets can be modeled with useful fidelity. Furthermore, Leithinger et al. (2011) found that only a very limited range of touch interactions could be recognized at the same time and that it can be challenging to manipulate individual pins as they may be out of reach. They augment touch with gestures by adding a Kinect as a depth camera so that users can easily change the context and explore larger datasets. While gestures are less precise than direct physical manipulation, they greatly expand the scope of possible interactions (Blackshaw et al., 2011). Interactions via external devices such as a mouse may be less ambiguous than gestures, but Leithinger et al. argue that they draw users' focus away from the shape display. Therefore they choose to combine touch interactions with gestures rather than pointing devices so that the transition from sculpting to selection, translation, rotation, and scaling would be fluid and seamless given the physical directness of both modes of interaction (Leithinger et al., 2011).

Tangible CityScape Tangible CityScape, a system built upon Recompose, is an example of how this type of TUI can be applied to a specific domain – urban planning. It used a 2.5D dynamic shape display to model and study urban massing. Building masses were modeled by clusters of pins and the

model dynamically reshaped as users panned or zoomed with gestures (Tang et al., 2013).

inFORM With inFORM Follmer et al. (2013) developed a dynamically changing user interface capable of diverse, rich user interactions. Building on the Relief and Recompose systems, they developed a 2.5D actuated shape display that supports object tracking, visualization via projection, and both direct and indirect physical manipulation. The shape display is moved by a dense array of pins linked by connecting rods to a larger array of actuators below (Fig. 1.3). The pins, pushing and pulling with variable pressure, offer nuanced haptic feedback to users. A Kinect depth sensor is used to track objects and users' hands. The actuated surface – a grid of pins – can be manipulated directly by pushing and pulling pins. Furthermore, users can interact with the system indirectly via object tracking. The surface can respond to interactions – both direct and indirect – and reshape itself. The actuated surface can also move objects placed on it, enabling indirect physical manipulations. Follmer et al. used this system to explore how shape-changing displays can dynamically model content and offer novel modes of interaction based on dynamically changing constraints and opportunities. As a responsive physical user interface inFORM enables rich and varied mode of interaction such as responsive sculpting, moving passive objects, painting changes, and physically instantiated UI elements like buttons (Follmer et al., 2013).

Dynamic shape displays that can be shaped both by touch and computation enable physical interactions from human to computer and computer to human. With a dynamic shape display the digital model is physically instantiated and this shape changing physical model is used for both input and output. When a user changes the surface of the shape display the computer reads the changes; when the computer changes the surface the user sees and feels the changes. Hypothetically this should radically reduce the level of abstraction in human-computer interaction. This novel mode of bidirectional, tangible interaction should be highly intuitive because it is so direct – human and computers communicate via the same medium. However, while dynamic shape displays may be highly intuitive, they have a relatively coarse resolution, are expensive, maintenance intensive, and hard to transport, all due to the actuators. The resolution of the display for example is constrained by the size of the actuators that make the display kinetic. The coarse resolution makes these displays' representations approximate and abstract, limiting their possible applications.

1.1.3 Continuous shape displays

Continuous shape displays couple a continuous physical model with a digital model through a cycle of sculpting, 3D scanning, computation, and projection. We can sculpt the display and then see the resulting computation projected onto the model in near real-time. The feedback from digital modeling

Fig. 1.3: inFORM: a 2.5D actuated shape display designed by the MIT Media Lab (image source: `http://tangible.media.mit.edu/project/inform/`)

is visual – unlike dynamic shape displays, these shape display are passive and do not physically respond to computations. Because there are no actuators the model can be built of a malleable, continuous material such as sand or clay that can be easily sculpted and can accurately represent forms such as landscapes (Ishii et al., 2004; Ishii, 2008b).

Projection Augmented Relief Models Currently continuous shape displays rely on projection for representing digital data. While projected imagery has long been used to augment physical terrain models the approach has recently been dubbed Projection Augmented Relief Models (Priestnall et al.,

2012). This approach enhances a static physical representation of a landscape with a dynamic visual representation of digital data.

Illuminating Clay With Illuminating Clay (Piper et al., 2002a) – an early continuous shape display from the MIT Media Lab – a clay model of a landscape was continuously scanned with a laser to generate a point cloud of x, y, and z coordinates which were then binned into a digital elevation model (DEM). The DEM or a derived topographic parameter such as slope, aspect, or cast shadow was then projected back onto the clay model so that users could see the impact of their changes in near real-time. Illuminating Clay's library of geospatial analyses was limited as it used custom implementations of elevation, aspect, slope, cast shadow, profile, curvature, viewshed, solar irradiation, and drain direction algorithms. Because Illuminating Clay used a laser scanner the scans were relatively fast – 1.2 seconds each – and accurate to less than 1 mm, but the system was very expensive (Piper et al., 2002a,b). Piper et al. (2002a) also tried to use TUI elements based on object tracking to select algorithms and parameters, but found that too many parameters were involved in geospatial modeling for this approach to be feasible, so they ultimately used standard graphical user interactions with a mouse and keyboard.

Sandscape In an effort to make a more affordable continuous shape display Ishii et al. (2004) designed a system called Sandscape based on infrared depth sensing. In Sandscape a 'sandbox' filled with 1 mm glass beads was lit from beneath by infrared LEDs; an infrared camera above the 'sandbox' captured the intensity of infrared light passing through the glass beads. A DEM based on the light intensity was projected back onto the 'sandbox' so that users would have real-time feedback as they sculpted. While this system was very fast and much less expensive than Illuminating Clay, the scans were low resolution and the physical medium – glass beads – could only be coarsely sculpted (Ishii et al., 2004; Ishii, 2008b).

Augmented Reality Sandbox SandyStation (Altman and Eckstein, 2014), developed in 2011, and the Augmented Reality Sandbox (Kreylos, 2015), developed in 2012, are continuous shape display sandboxes. Kreylos's Augmented Reality Sandbox, inspired by the Czech SandyStation, couples a sandbox with a DEM through a real-time cycle of 3D scanning, modeling and simulation, and projection. As users sculpt the sand a Kinect sensor continually scans the sand surface generating a stream of depth maps. The depth stream is filtered to remove noise and the user's hands. A DEM, contours, and simulated water flow are computed from the depth and projected back on the sandbox. The water flow simulation is a custom implementation based of the shallow water flow equations. The Augmented Reality Sandbox is powered by Kreylos's Kinect 3D Video Capture Project and Vrui VR Toolkit, open source software released under the GNU General Public License. The

system is highly intuitive and very fast with each cycle of scanning, processing, and simulation completing in just a couple seconds. However, given that the system only models elevation, contours, and water flow on an abstract landscape it is limited in its scope of potential applications.

Tangible Geospatial Modeling System The Tangible Geospatial Modeling System (TanGeoMS) extended the concept of Illuminating Clay by integrating GRASS GIS (Fig. 1.4). Integrating a full GIS into a continuous shape display gave users a wide range of sophisticated, scientific tools for geospatial modeling and visualization in a highly intuitive environment. By combining freeform hand modeling with geospatial modeling, simulation, and visualization users could easily explore how changes in topographic form affect landscape processes (Mitasova et al., 2006). Like Illuminating Clay the system used a laser scanner to 3D scan a clay model. The scanned point cloud was then interpolated as a DEM, select geospatial analyses or simulations were computed from the DEM, and the results were projected back onto the clay model (Tateosian et al., 2010) (Fig. 1.4b).

1.2 Tangible Landscape

The next generation of TanGeoMS – Tangible Landscape – was designed to intuitively 3D sketch landscapes. This continuous shape display is more tightly integrated with GRASS GIS and has new modes of interaction based on object recognition so that users cannot only sculpt, but also sketch in 3D – rapidly exploring ideas while learning from near real-time feedback (Fig. 1.5). Tangible Landscape is more affordable than its predecessors because it uses a Kinect sensor to 3D scan physical models. As a continuous shape display with object recognition a wide range of media can be used such as polymer-enriched sand, clay, 3D prints, CNC-machined models, and architectural models. It is more tightly integrated with GIS, using a GRASS GIS plugin to automatically scan, process, georeference, import, and analyze the model. Because it is so tightly integrated with GRASS GIS users can also use the GUI, the command line, and scripting as advanced controls for tasks not suited to a TUI (Petrasova et al., 2014).

Conceptually, Tangible Landscape couples a physical model with a digital model in a real-time feedback cycle of 3D scanning, geospatial modeling and simulation, and projection. For example, by sculpting the terrain of the physical model, we can see how the changes affect processes like the flow of water, flooding, erosion, and solar irradiation. Thus we can easily and rapidly test ideas while being guided by scientific feedback, exploring a much a larger solution space and make more creative and informed decisions. And since many users can interact with the physical model at once, Tangible Landscape encourages collaboration and interdisciplinary exchange (Fig. 1.6). Tangible Landscape combines real-time interaction with extensive scientific tools for

(a) Multiple users interacting

(b) The shape display paired with a 3D rendering

(c) The hardware setup with a laser scanner mounted above the model

Fig. 1.4: The Tangible Geospatial Modeling System: a continuous shape display powered by GRASS GIS

(a) Orthoimagery

(b) Water flow

(c) Sculpting a check dam

(d) Updated water flow

(e) A dammed valley with water flow simulated with *r.sim.water*

Fig. 1.5: 3D sketching a check dam with Tangible Landscape

modeling, analysis, and visualization at the precision needed for real-world design and planning applications. Tangible Landscape can be used for applications such as grading landforms (chapter 3), analyzing topography (chapter 4), modeling water flow and soil erosion (chapter 5), analyzing viewsheds (chapter 6), planning trail networks (chapter 7), analyzing solar radiation dynamics (chapter 8), simulating and managing fire (chapter 9), and modeling inundation and flooding (chapter 10).

Tangible Landscape is inexpensive, portable, adaptable, and easy to implement. As a high resolution continuous shape display it can be used for both rapid ideation and precise modeling. However, unlike a dynamic shape display the physical model is passive and cannot be changed computationally in real-time – trading real-time response for precision, complexity, and cost. Rapid prototyping can be used to precisely shape or reshape the physical model allowing users to work with real world landscapes and easily reset the conditions.

Fig. 1.6: Collaboratively sculpting a retention pond

1.2.1 Modes of interaction

There are many ways to 3D sketch with Tangible Landscape. It combines a couple modes of tangible user interaction. As a continuous shape display – a 2.5D interface that physically instantiates and displays continuous data in real-time – we can use it sculpt surfaces and volumes. It can also use object recognition to detect and compare changes, model volumes, designate points, select data, and sketch lines. By sculpting a physical model made of

polymeric sand we can intuitively model topography (Fig. 1.7), forest canopy, buildings, and cost surfaces. We can also explore volumes such as subsurface soil moisture by digging the into polymeric sand. With change detection we can visualize the difference between models to study how we have changed it. With object recognition we can place makers to create points such as observer points for viewshed analysis, lines such as section cuts for visualizing volumes, and polylines such as the least cost path between points for route analysis. Since Tangible Landscape has so many modes of interaction and draws on GRASS GIS' extensive modules for data management, geospatial modeling, and visualization, it can easily be adapted to new uses. Since both Tangible Landscape and GRASS GIS are open source projects we can easily develop new modes of interaction and new analyses.

(a) A flooded landscape (b) Breaching the levee (c) Updated flooding

Fig. 1.7: 3D sketching a levee breach with feedback on the resulting extent of flooding

1.2.2 Developing Tangible Landscape

Tangible Landscape is a free, open source project with source code hosted on GitHub.[1] The system is constantly evolving in response to new developments in its hardware and software components and the needs of its expanding range of applications. Enhanced point cloud processing, optimization of the core functions, and migration to more efficient hardware and operating systems will improve real-time interaction and allow us to incorporate more sophisticated dynamic modeling. User interaction and experience can be further improved with dashboards and steered simulations. Computationally controlled shape generation with in-situ digital fabrication will combine the advantages of dynamic and continuous shape displays. By providing Tangible Landscape as a free, open source project, along with this book, we aim to build a community of developers and users of this technology for the benefit of education, research and collaborative decision making in communities worldwide.

[1] `https://github.com/ncsu-osgeorel/grass-tangible-landscape`

1.3 The organization of this book

This book describes tangible geospatial modeling using open source GIS and its applications. For each application we explain the underlying theory and algorithms, provide the workflows used, and describe case studies. Simple GRASS GIS workflows consisting of module calls are presented in boxes with green backgrounds, while more complex workflows written in Python are presented in boxes with blue backgrounds. 3D modeling workflows in Rhinoceros are presented in boxes with grey backgrounds.

Chapter 2 describes how Tangible Landscape works and how it is set up. This chapter details the 3D scanning technology, the hardware and software used, and the physical setup of the system. It provides information about where to download GRASS GIS and the Tangible Landscape plugin and explains how to read and use the workflows. Chapter 3 explains how to build physical models for use with Tangible Landscape. This chapter covers handmade modeling techniques, digital fabrication techniques, and techniques for molding and casting models. Chapter 4 covers the basic processes of scanning, interpolating, and analyzing a terrain model. The following chapters describe a range of scientific applications for Tangible Landscape. Each chapter first covers the relevant theory and algorithms and then demonstrates the application in a case study. Chapter 5 covers surface water flow and soil erosion, chapter 6 covers viewsheds, chapter 7 covers trail planning, chapter 8 covers solar irradiation dynamics, chapter 9 covers the spread of wildfire, and chapter 10 covers coastal inundation and flooding. The Appendix includes case studies of additional applications of Tangible Landscape, a summary of the data used in this book with download links, a list of useful on-line data sources, and a guide to getting started with GRASS GIS.

References

Altman, P. and Eckstein, R. SandyStation [online]. (2014) [Accessed 2015-08-12]. `http://en.sandystation.cz/`.

Blackshaw, M., DeVincenzi, A., Lakatos, D., Leithinger, D., and Ishii, H. (2011). Recompose: direct and gestural interaction with an actuated surface. In *Proceedings of the 2011 annual conference extended abstracts on Human factors in computing systems - CHI EA '11*, page 1237, Vancouver. ACM Press.

Coelho, M. and Zigelbaum, J. (2010). Shape-changing interfaces. *Personal and Ubiquitous Computing*, 15(2):161–173.

Dourish, P. (2001). *Where the action is : the foundations of embodied interaction*. MIT Press, Cambridge, MA.

Fitzmaurice, G. W., Ishii, H., and Buxton, W. (1995). Bricks: laying the foundations for graspable user interfaces. *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 442–449.

Follmer, S., Leithinger, D., Olwal, A., Hogge, A., and Ishii, H. (2013). inFORM: dynamic physical affordances and constraints through shape and object actuation. In *UIST '13 Proceedings of the 26th annual ACM symposium on User interface software and technology*, pages 417–426, St. Andrews, UK. ACM Press.

Goulthorpe, M. Aegis Hyposurface [online]. (2000) [Accessed 2015-04-10]. `http://hyposurface.org/`.

Ishii, H. (2008a). Tangible bits: beyond pixels. In *Proceedings of the 2nd international conference on Tangible and embedded interaction - TEI '08*, pages xv–xxv, Bonn, Germany. ACM Press.

Ishii, H. (2008b). The tangible user interface and its evolution. *Communications of the ACM*, 51(6):32–36.

Ishii, H., Ratti, C., Piper, B., Wang, Y., Biderman, a., and Ben-Joseph, E. (2004). Bringing Clay and Sand into Digital Design Continuous Tangible user Interfaces. {*BT*} *Technology Journal*, 22(4):287–299.

Ishii, H. and Ullmer, B. (1997). Tangible bits: towards seamless interfaces between people, bits and atoms. In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '97*, pages 234–241, New York, New York, USA. ACM Press.

Iwata, Yano, Nakaizumi, and Kawamura (2001). Project FEELEX: adding haptic surface to graphics. *Proceedings of SIGGRAPH 2001*, pages 469–475.

Kreylos, O. Augmented Reality Sandbox [online]. (2015) [Accessed 2015-08-12]. `http://idav.ucdavis.edu/~okreylos/ResDev/SARndbox/index.html`.

Leithinger, D. and Ishii, H. (2010). Relief: a scalable actuated shape display. In *Proceedings of the fourth international conference on Tangible, embedded, and embodied interaction - TEI '10*, page 221, Cambridge, MA. ACM Press.

Leithinger, D., Kumpf, A., and Ishii, H. Relief [online]. (2009). `http://tangible.media.mit.edu/project/relief/`.

Leithinger, D., Lakatos, D., Devincenzi, A., Blackshaw, M., and Ishii, H. (2011). Direct and Gestural Interaction with Relief : A 2 . 5D Shape Display. *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 541–548.

Mitasova, H., Mitas, L., Ratti, C., Ishii, H., Alonso, J., and Harmon, R. S. (2006). Real-time landscape model interaction using a tangible geospatial modeling environment. *IEEE Computer Graphics and Applications*, 26(4):55–63.

Petrasova, A., Harmon, B. A., Petras, V., and Mitasova, H. (2014). GIS-based environmental modeling with tangible interaction and dynamic visualization. In Ames, D. and Quinn, N., editors, *Proceedings of the 7th International Congress on Environmental Modelling and Software*, San Diego, California, USA. International Environmental Modelling and Software Society.

Petrie, G. (2006). TouchTable and TerrainTable. *Geoinformatics*, pages 40–41.

Piper, B., Ratti, C., and Ishii, H. (2002a). Illuminating clay: a 3-D tangible interface for landscape analysis. In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '02*, page 355, Minneapolis. ACM Press.

Piper, B., Ratti, C., and Ishii, H. (2002b). Illuminating Clay: A Tangible Interface with potential GRASS applications. In *Proceedings of the Open Source GIS - GRASS users conference 2002*, Trento, Italy.

Poupyrev, I., Nashida, T., and Okabe, M. (2007). Actuation and tangible user interfaces: the Vaucanson duck, robots, and shape displays. *Proceedings of TEI 2007*, pages 205–212.

Poynor, R. (1995). The Hand That Rocks the Cradle. *ID Magazine*, pages 60–65.

Priestnall, G., Gardiner, J., Way, P., Durrant, J., and Goulding, J. (2012). Projection Augmented Relief Models (PARM): Tangible Displays for Geographic Information. *Proceedings of Electronic Visualisation and the Arts, London*, pages 1–8.

Tang, S. K., Sekikawa, Y., Leithinger, D., Follmer, S., and Ishii, H. Tangible CityScape [online]. (2013) [Accessed 2014-03-27]. `http://tangible.media.mit.edu/project/tangible-cityscape/`.

Tateosian, L., Mitasova, H., Harmon, B. A., Fogleman, B., Weaver, K., and Harmon, R. S. (2010). TanGeoMS: tangible geospatial modeling system. *IEEE transactions on visualization and computer graphics*, 16(6):1605–12.

Chapter 2
System configuration

The setup of the Tangible Landscape system consists of four primary components: (a) a physical model that can be modified by a user, (b) a 3D scanner, (c) a projector, and (d) a computer installed with GRASS GIS for geospatial modeling and additional software that connects all the components together. The physical model, placed on a table, is scanned by the 3D scanner mounted above. The scan is then imported into GRASS GIS, where a DEM is created. The DEM is then used to compute selected geospatial analyses. The resulting image or animation is projected directly onto the modified physical model so that the results are put into the context of the modifications to the model.

2.1 Hardware

Tangible Landscape can be built with affordable, commonly available hardware: a 3D scanner, a projector and a computer. We describe some of the current hardware options while noting that the technology develops rapidly and alternative, more effective solutions may emerge.

2.1.1 3D scanner

In the Tangible Landscape workflow the 3D scanner captures the physical model as it is modified by users. Therefore it is important that the device scans as accurately as the technology allows. Only few affordable 3D scanners are currently available. However, due to the growing demand this technology is developing rapidly and new, improved 3D scanners may become available soon. The Tangible Landscape system was developed using the Kinect for Windows v2 and thus has the disadvantage of being bound to the Microsoft Windows operating system. Being bound to a specific operating system limits the choice of environment in which to run the application. Therefore, the selection of the 3D scanner may change in the future as new, more flexible technologies become available.

The technology behind 3D scanning We describe the basic principles behind current 3D depth sensing technologies in order to explain both the potential and the limitations of current scanning devices. More detailed information can be found for example in Mutto et al. (2012).

Several currently available depth sensors such as Apple Primesense Carmine, Asus Xtion PRO LIVE, [1] and Kinect for Windows v1, are based on a Primesense proprietary light-coding technique. This technique uses triangulation to map 3D space in manner similar to how the human visual system senses depth from two slightly different images. Rather than using two cameras it triangulates between a near-infrared camera and a near-infrared laser source. Corresponding objects need to be identified in order to triangulate between the images. A light coding (or structured light) technique is used to identify the objects. The laser produces a pseudo-random dot pattern which is then used to find the matching dot pattern in the projected pattern. In this way the final depth image can be constructed.

Optical triangulation is also used by high resolution 3D scanners such as SCANIFY by Fuel3D (2015). The scanner acquires a series of stereoscopic photographs with different lighting directions. When scanning multi-color objects with some texture this technology can create highly accurate 3D models with high levels of detail. The major limitation is that only relatively small models that can be scanned making it unsuitable for most applications using physical landscape models. These models usually need to be relatively large to clearly represent relief at a landscape scale without too much vertical exaggeration.

Kinect for Windows v2 uses *time-of-flight*, a technique widely used in lidar technology. Kinect v2 has a sensor that indirectly measures the time it takes for pulses of near-infrared laser light to travel from a laser source, to a target surface, and then back to the image sensor. Time-of-flight sensors are generally considered to be more precise, but also more expensive.

Since the scanning and 3D modeling algorithms and their implementations for most sensors are proprietary, the specific behavior and precision of particular sensors is subject to many experimental studies. Generally, sensors using near-infrared light are sensitive to lighting conditions, so outdoor usage is typically not recommended. The depth resolution decreases with the distance from the sensor and also at close range (i.e. between a couple of millimeters to tens of centimeters). The range and field of view of the sensors can vary; Tangible Landscape requires short range sensors with a minimum distance of 0.5 m to keep the highest possible resolution. When scanning with one sensor the size of the physical model is limited by the required accuracy because the sensor must be far enough away to capture the entire model in the sensor's field of view, which for the Kinect for Windows v2 is 60° vertical by 70° horizontal. Detailed information about accuracy and precision of Kinect for Windows v1 and v2 can be found in Andersen et al. (2012); Lachat

[1] `https://www.asus.com/us/Multimedia/Xtion_PRO_LIVE/`

et al. (2015) and a comparison to Asus Xtion can be found in Gonzalez-Jorge et al. (2013).

2.1.2 Projector

The projector projects the background geospatial data and results of analyses onto the the 3D physical model. Therefore it is important to select a projector with sufficient resolution and properties that minimize distortion and generate a bright image.

Resolution and brightness are two important criteria to be considered. We recommend higher resolution projectors offering at least WXGA (1280×800). The brightness depends upon where Tangible Landscape is used and whether the room's ambient light can be reduced for the sake of brighter projected colors. We recommend brighter projectors (at least 3000 lumens) since we project on a variety of materials which are not always white and reflective.

There are two important types of projectors – standard and short-throw projectors. They differ in throw ratio values, which are defined as the distance from the projector's lens to the screen, divided by the width of the projected image (short throw and ultra short throw projectors have ratios 0.3 - 0.7, while the standard projectors have throw ratio values around 2). At the same distance the projector with a lower throw ratio can display a larger image. In other words a projector with a lower throw ratio (short-throw) can project an image of the same size as the higher throw ratio projector, but from a shorter distance. Tangible Landscape can be set up using both types of projectors; each has advantages and disadvantages.

The placement and configuration of the projector is important because it affects the coverage, distortion, and visibility of the projected data. For example in some setups the 3D scanner may be caught in the projection and would thus cast a shadow over the model. Therefore it can be practical to use a short-throw projector because it can be placed to the side of the physical model at a height similar to the 3D scanner (Fig. 2.1b and 2.2). Since the projection is cast from the side the projection beam does not cross the 3D scanning device and no shadow is cast. However, with a short-throw projector there is a certain level of visible distortion when projecting on a physical model that has substantial relief. The distortion occurs because the light rays reach the model at a very acute angle. The horizontal position at which the projected light intersects the model is shifted from the position at which it would intersect with a flat surface. Larger differences in height increase the distortion. Theoretically we can remove the distortion by either warping the projected data itself or using the projector to automatically warp the projected image. The first solution would require an undistorted dataset for geospatial modeling and a warped copy of that data for projecting. That is impractical especially when working with many different raster and vector layers. The other solution requires the projector itself to warp the image;

while this technology exists, it is only offered by a few projector manufacturers and such projectors are typically more expensive. Moreover, as the landscape is modified, the warping pattern should change as well. Currently it is not possible to find this feature in the off-the-shelf projectors.

With standard projectors the distortion is usually negligible since the incidence angle of the rays is relatively small. A standard projector needs to be placed much higher above the model than a short-throw projector does due to the difference in throw ratios (Fig. 2.1a and 2.3). This creates several challenges. It is hard to mount and manipulate the projector when is it so high above the model. Furthermore it is challenging to align projector and the 3D scanner without casting a shadow. Since the 3D scanner has a limited field of view it must be placed close to the horizontal center of the model. When the projector is mounted above the scanner, the scanner is caught in the projector's beam, casting a shadow over part of the model. For small to mid sized models, depending on the particular setup and the height of the scanner and projector, this may not be a problem.

The shadow of the scanner can also be avoided with specialized short throw projectors that allow greater installation flexibility through lens shifting. These projectors (for example Canon WUX400ST) are capable of projecting from the horizontal center of the projected area at the same height as the scanner (Fig. 2.1c). A device which combines the scanner and projector would make this setup easy; an appropriate device, however, was not available at the time of writing.

To minimize the distortion when using projectors with lower throw ratios, we can project from the center by tilting the projector and correcting the resulting keystone distortion (Fig. 2.1d). With this setup we recommend projectors that have throw ratios around 1.0 (for example Epson PowerLite 1700 Series) as a lower ratio can result in additional distortion. When testing the projector setup it is useful to project the rectangular grid on a flat surface in order to quickly check if there is any distortion.

2.1.3 Computer requirements

System requirements depend largely upon the sensor and its associated library or software development kit (SDK). Certain sensors, such as Kinect for Windows, are designed to work only on specific operating systems. The computer should be configured for 3D scanning and geospatial modeling, both of which are performance- and memory-intensive processes. The hardware requirements are very similar to the requirements for gaming computers: a multi-core processor, at least 4 GB of system memory and a good graphics card are necessary to achieve real-time interaction with the model. The specific parameters required for the scanner device should be reviewed on the manufacturer's website.

(a) Standard projector mounted in the ceiling

(b) Standard short throw projector

(c) Short throw projector capable of projecting above the center

(d) Short throw projector tilted to project above the center with keystone correction

Fig. 2.1: Possible placements of the projector and scanner according to projector type

2.1.4 Physical setup

The setup for Tangible Landscape system is quite flexible. However, there are some crucial components and some specifics that make a setup more usable. When building Tangible Landscape in the laboratory or when bringing it to the community the following items are necessary:

- a table for a model
- a laptop or desktop computer with a table
- a scanner
- a projector
- 1-2 stands for a projector and a scanner
- 3-4 power plugs (and/or extension cable)

Ideally the table should be either a 90 cm $\times$ 90 cm square table with rounded corners, a rounded table 100 cm in diameter, or a teardrop table of similar size. To freely interact with the model a person should be able to almost reach the other side of the table; this is not possible with larger tables. Smaller tables, on the other hand, have less space for tools, additional sand, and application windows. Ideally application windows with additional information should be projected onto the table beside the model. A large model can be placed on top of a smaller table or stand, provided that this base is stable. In this case if any application windows are needed, they have to be projected directly onto the model.

A round table is quite advantageous because people can stand at equal distances from the model and can easily move around the table. Unfortunately, one side of the table is typically occupied by the metal stands for projector and scanner since ceiling mounts are rarely possible and render the system immobile. The table should be stable enough to hold sand and models and sturdy enough to withstand their manipulation. We recommend putting the computer on a separate table so that the modeling table is not cluttered. The more space there is around the modeling table, the more access users have to the model. The computer, however, needs to be situated so that its operator has easy access to the model as well.

An alternative is to use a rectangular table (ideally with rounded corners) that is 140 cm $\times$ 90 cm for both the model and the laptop. A narrow table that is 150 cm $\times$ 75 cm may work as well. This setup makes the model less accessible as at least two sides of the model are blocked. However, the whole model should be accessible from any of the remaining sides, so this setup does not necessarily limit interaction, but rather the number of users.

Sometimes it is advantageous to have a large screen showing additional data or 3D views to enhance users' understanding of the processes. However, such screen should not limit access to the physical model. Tangible Landscape could be also extended with additional devices such as 3D displays, head-mounted displays, and hand-held devices.

There should be at least 80 cm of space on each side of the modeling table so that there is room for walking, standing, and placing the stands. The whole area required is about 2.5 m $\times$ 2.5 m but in practice the area is often larger depending on the size of the tables and the surroundings. Generally, it is better to have a larger open space. With more space people can easily come and cluster around Tangible Landscape spreading around the different sides of the setup.

Fig. 2.2: Mobile system setup: projector, scanner, model and laptop

The projector and scanner can be mounted on one or two stands according to projector's capabilities as discussed in detail in Section 2.1.2. Two separate stands or arms give enough flexibility to accommodate various types of projectors. In any case, the stand for the projector must be robust because some of the projectors with adequate parameters can be heavy. However, there exist some portable projectors weighting less than 2 kg as well. A mobile setup with two separate stands and two separate tables is shown in Fig. 2.2. For permanent setups the projector can be mounted in the ceiling as shown in Fig. 2.3.

Fig. 2.3: Laboratory system setup with the projector mounted in the ceiling

2.2 Software

The software for Tangible Landscape consists of several different components: a scanning program dependent upon the Kinect for Windows SDK libraries, GRASS GIS for processing scans and running geospatial analyses, and a custom library that automates the entire process. Here we briefly describe the software components with the main focus on GRASS GIS as the processing

engine of Tangible Landscape. Tangible Landscape software is available in the GitHub repository[2] under the GNU GPL license.

2.2.1 GRASS GIS

GRASS GIS (Neteler and Mitasova, 2008) is a general purpose cross-platform, open-source geographic information system with raster, vector, 3D raster and image processing capabilities. It includes more than 400 modules for managing and analyzing geographical data and many more user contributed modules available in the add-on repository. GRASS GIS modules can be run using a command-line interface (CLI) or a native graphical user interface (GUI) called wxGUI which offers a seamless combination of GUI and CLI native to the operating system.

Modules are organized based on the data type they handle and they follow naming conventions explained in Table 2.1. Each module has a set of defined options and flags which are used to specify inputs, outputs, or different module settings. Most core modules using GRASS GIS C library are written in C for performance and portability reasons. Other modules and user scripts are written in Python using the Python Scripting Library which provides a Python interface to GRASS GIS modules.

Name	**Data type**	**Examples**
g.*	general data management	g.list, g.remove, g.manual
r.*	raster data	r.neighbors, r.viewshed, r.cost
r3.*	3D raster data	r3.colors, r3.to.rast, r3.cross.rast
v.*	vector data	v.net, v.surf.rst, v.generalize
db.*	attribute data	db.tables, db.select, db.dropcolumn
t.*	temporal data	t.register, t.rast.aggregate, t.vect.extract
i.*	imagery data	i.segment, i.maxlik, i.pca

Table 2.1: Naming conventions for GRASS GIS modules with examples

GRASS GIS software can be downloaded freely from the main GRASS project website.[3] The download web page offers easy to install binary packages for GNU/Linux, Mac OS X, and Microsoft Windows as well as the source code. The GRASS GIS website also provides additional documentation including manual pages, tutorials, information about the externally developed modules (add-ons) and various publications. Support for developers and users is provided by several mailing lists. The following tutorial provides a quick introduction to GRASS GIS: `http://grass.osgeo.org/grass70/manuals/helptext.html`.

[2] `https://github.com/ncsu-osgeorel/grass-tangible-landscape`

[3] `http://grass.osgeo.org`

2.2.2 GRASS GIS Python API

GRASS GIS provides several Python application programming interfaces (API) to access different functionalities and accommodate many different use cases. Tangible Landscape uses the Python Scripting Library to easily build new workflows by chaining together existing GRASS GIS modules. We will show an example how to use this library to automate tasks and build new functionality.

GRASS GIS modules are called with the following command syntax:

$$\underbrace{\texttt{r.colors}}_{\text{module name}}\ \ \underbrace{\texttt{-e}}_{\text{flag}}\ \ \underbrace{\texttt{raster}}_{\text{option name}}\texttt{=}\underbrace{\texttt{elev_state_500m}}_{\text{option value}}\ \ \texttt{color=elevation}$$

This command assigns a predefined color ramp `elevation` to the raster map `elev_state_500m`. It can be executed in a GRASS GIS session from a terminal or from a GRASS GIS GUI command console. The same command looks very similar when written using the Python Scripting Library – it is just adjusted to Python syntax. To run it we have to be in a GRASS GIS session and must first import the necessary library.[4] We will use function `run_command` to call individual modules:

```
import grass.script as gscript
gscript.run_command('r.colors', flags='e',
    raster='elev_state_500m', color='elevation')
```

Besides `run_command`, other important functions are `read_command` for reading the raw module output, `parse_command` for reading module output already parsed into a Python dictionary, and `write_command` for passing strings from standard input. All GRASS GIS modules can be called using these functions. However, for some commonly used modules the library conveniently provides wrappers simplifying the syntax, such as the `mapcalc` function, a wrapper for the *r.mapcalc* module.

The following code snippet provides a more complex example using these functions. Here, we extract low-lying areas from a DEM where the elevation z is lower than the difference $z_{mean} - z_{stddev}$:

```
import grass.script as gscript

def main():
   input_raster = 'elevation'
   output_raster = 'low_areas'
   stats = gscript.parse_command('r.univar', map=input_raster,
       flags='g')
   mean = float(stats['mean'])
   stddev = float(stats['stddev'])
   low = mean - stddev
```

[4] All further Python code snippets assume the library is already imported to avoid code duplication.

```
    gscript.mapcalc('{out} = {inp} <
        {lim}'.format(out=output_raster, inp=input_raster,
        lim=low))

if __name__ == "__main__":
    main()
```

More information on writing Python scripts in GRASS GIS can be found in the online Python Scripting Library documentation.[5]

2.2.3 Scanning component

The scanning component is currently a C++ application derived from the sample application KinectFusionExplorer shipped with the Kinect for Windows SDK. It reconstructs a mesh based on multiple scans using the KinectFusion algorithm (Newcombe et al., 2011) and writes the point cloud as a text file. It allows us to set the scanning properties such as the resolution of the reconstructed mesh and to limit the volume of the mesh. We recommend using a higher volume resolution such as 512 or 640 voxels per meter. It is important to make the volume of the mesh as small as possible to increase processing speed and decrease memory demands. Since the size of the mesh can only be controlled in predefined steps we cannot limit the scanned area to the model area in this way – further processing of the point cloud to filter the points outside of the model area is needed (Fig. 2.4).

2.2.4 Tangible Landscape plugin for GRASS GIS

The Tangible Landscape plugin connects the scanning component with GRASS GIS and automates the loop of scanning, importing scans, and geoprocessing in the GRASS GIS environment. It has a graphical user interface which allows the adjustment of the different processing parameters that are necessary to properly georeference and extract the model. First it can perform an initial calibration. Then with each new scan the plugin performs the following basic steps:

1. acquiring the scan as a point cloud,
2. extracting only relevant points from the point cloud (filtering points outside the model area)
3. georeferencing (horizontal rotation, horizontal and vertical scaling, translation) the scanned data to known geographic coordinates on the edges of the model,
4. importing the point cloud into GRASS GIS and creating a digital elevation model,
5. running selected geospatial analyses,

[5] Python Scripting Library documentation: `http://grass.osgeo.org/grass70/manuals/libpython/script_intro.html`

Fig. 2.4: A mesh with a resolution of 512 voxels per meter constructed using the KinectFusion algorithm. The mesh captures a sand model, the area outside of the model including a pile of sand and a modeling tool, and the hands of a person interacting with the model.

6. and re-rendering the resulting raster or vector data that are projected on top of the model.

The conversion from the scanned point cloud to a DEM is an automated process; the details and assumptions that have to be met to produce good results are described in the following paragraphs.

Calibration During scanning the scanner axis should be oriented precisely perpendicular to the table with the model in order to avoid a tilted scan. Even a small deviation of 1 degree can cause centimeters of height difference depending on the size of the model (see Fig. 2.5a). Therefore, we first calibrate the system by scanning the empty table and computing the angular deviation. We then use this information to automatically rotate each scanned point cloud.

From the scan of the empty table, we compute the plane that best fits the scan by applying the linear regression in 3D. We compute the angular deviation between the unit vector along the scanner axis and the vector perpendicular to the plane. We then use Rodrigues' rotation formula (Wikipedia, 2015) to derive a rotation matrix to rotate the point cloud by this angular deviation around a vector given as a cross product of the two vectors. After calibration, the derived rotation matrix is used for rotating every scan resulting in perfectly horizontal scan, see Fig. 2.5b. If the scanner is moved the calibration must be repeated.

For more precise results, we can also minimize the radial distortion of the scanner visible in Fig. 2.5b. The simplest way to do this is to acquire a scan

of a flat surface with the dimensions of the physical model and then subtract it from the raster representation of the model.

(a) (b)

Fig. 2.5: Even slight angular deviation from the vertical scanner axis will cause a tilted scan as seen in (a) on a scan of a flat plane with a small angular deviation of 1.7°. A calibration based on the initial scan of a flat plane can compensate for such angular deviations resulting in a horizontally aligned scan as shown in (b) although a small radial distortion will still be present

.

Model extraction Since the size of the scanned area cannot be easily limited to just the physical model (see Fig. 2.4), we need to find the boundaries of the model in order to georeference the scanned data.

The first part of the process is to identify points that represent the underlying table. These points have very similar (but due to noise nonidentical) low values of z coordinates. If enough of the table is captured in the scan, we can find the interval of z coordinates containing the points of the table as a first local maximum in a histogram of z values. These points can then be discarded.

After removing these points, we attempt to remove points outside of the model area such as leftover sand or arms reaching over the model in Fig. 2.4. This can be successfully done when the model is rectangular, since we can estimate the horizontal dimensions of the model from the scan and then trim the areas that do not contain enough of points from each side. If the model is not rectangular we can manually choose how many centimeters to trim from each side. Manual trimming, however, requires adjustments anytime that the model is moved even slightly.

Georeferencing Georeferencing the scanned model is an important step when we need to combine it with our geographic data and to ensure that any geospatial analyses are performed on a DEM with real-world dimensions. We need to designate the DEM raster map that the model represents and specify the vertical exaggeration of the physical model to scale the elevation values properly. The scale factors S_x, S_y and S_z are computed:

$$S_x = \frac{Y_{north} - Y_{south}}{y_{max} - y_{min}}, \quad S_y = \frac{X_{north} - X_{south}}{x_{max} - x_{min}}, \quad S_z = \frac{(S_x + S_y)/2}{e} \tag{2.1}$$

where X, Y are DEM (real-world) coordinates, x, y are coordinates on the physical model and e is the specified vertical exaggeration. The vertical scale of the model typically differs from the horizontal because we vertically exaggerate the physical model to enhance our perception of the landscape and simplify our interaction with the model. Since Tangible Landscape users usually interact with the physical model from the side opposite to the scanner we have to rotate the scan by 180 degrees. We can georeference the point cloud with the following equation:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & S_z \end{bmatrix} \cdot \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \tag{2.2}$$

which rotates the points around the z axes by angle α in the counterclockwise direction, scales to real-world dimensions, and translates the points by adding t_x and t_y computed so that the lower left corner of the model matches the south-west corner of the DEM. The vertical translation t_z is then similarly computed to match the lowest point of the model and the minimum height of the DEM.

DEM processing The plugin allows users to select the method for creating the DEM from the point cloud, which can be either binning or interpolation (see Section 4.1). Once the DEM is created all geospatial analyses specified in a dedicated Python file are sequentially performed. Each separate analysis is represented as one function. These functions are reloaded for each new processing loop so that we can easily run different algorithms and change their parameters. This flexible system enables us to dynamically develop and test new workflows during scanning. The plugin provides a library containing often used analyses and workflows which can be readily used and combined with other GRASS GIS functionality using the Python Scripting Library.

References

Andersen, M., Jensen, T., Lisouski, P., Mortensen, A., Hansen, M., Gregersen, T., and Ahrendt, P. (2012). Kinect Depth Sensor Evaluation for Computer Vision Applications. Technical report, Aarhus University, Department of Engineering. Denmark.

Fuel3D. SCANIFY: The low-cost, high-quality handheld 3D Scanner from Fuel3D [online]. (2015) [Accessed 2015-08-13]. `https://www.fuel-3d.com/`.

Gonzalez-Jorge, H., Riveiro, B., Vazquez-Fernandez, E., Martínez-Sánchez, J., and Arias, P. (2013). Metrological evaluation of Microsoft Kinect and Asus Xtion sensors. *Measurement*, 46(6):1800–1806.

Lachat, E., Macher, H., Mittet, M.-a., Landes, T., and Grussenmeyer, P. (2015). First Experiences With Kinect V2 Sensor for Close Range 3D Modelling. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL-5/W4(February):93–100.

Mutto, C., Zanuttigh, P., and Cortelazzo, G. (2012). Introduction. In *Time-of-Flight Cameras and Microsoft Kinect*, SpringerBriefs in Electrical and Computer Engineering, pages 1–15. Springer US.

Neteler, M. and Mitasova, H. (2008). *Open source GIS: a GRASS GIS approach*. New York: Springer, third edition.

Newcombe, R. a., Molyneaux, D., Kim, D., Davison, A. J., Shotton, J., Hodges, S., and Fitzgibbon, A. (2011). KinectFusion : Real-Time Dense Surface Mapping and Tracking. *IEEE International Symposium on Mixed and Augmented Reality*, pages 127–136.

Wikipedia. Rodrigues' rotation formula — wikipedia, the free encyclopedia [online]. (2015) [Accessed 2015-08-13]. `https://en.wikipedia.org/w/index.php?title=Rodrigues%27_rotation_formula&oldid=671556479`.

Chapter 3
Building physical 3D models

Fig. 3.1: A CNC routed model of the Sonoma Valley, California

Tangible Landscape works with many types of physical 3D models. When used as a continuous shape display the physical model should be built of a malleable material such as sand, clay, or wax so that users can easily deform the surface. When used for object recognition the physical model can be built of a rigid material such as a wood product, plastic, or resin. When both modes of interaction are combined the physical model should use malleable materials for the base and rigid materials for the objects. These models can be built by hand or digitally fabricated using using 3D printing or computer numerically controlled (CNC) manufacturing.

The final model should be opaque, have a light color, and have a matte finish so that the projected image is crisp and vivid. Transparent materials like acrylic cannot be 3D scanned with a Kinect. Some 3D printing and casting materials like resin may appear opaque, but have translucent properties – this will diffuse the projection. If we desire a very crisp and vivid image on

a rigid model made of wood products or resins we recommend painting the model white.

In this chapter we discuss different types of physical models and explain how to fabricate them.

3.1 Handmade models

Fig. 3.2: Sculpting a polymeric sand model

Rigid physical models can be handmade by hand-cutting contours and malleable physical models can be handmade by sculpting sand or clay.

Contour models Contour models can be precise if they are finely cut, but they are inaccurate as they depict abstract, stepped landscapes that are discrete rather than continuous. They are also very legible – one can easily count the contours and read the height – but again they represent abstracted landscapes. While one can easily read the contours and then calculate slope, it is hard to intuitively visualize the slope. Furthermore this abstract, discontinuous representation may obfuscate the relationship between form and process. Furthermore they are time consuming and complicated to construct especially for complex topographies. Hand cutting contour models can be dangerous as the knife may slip or jump.

Contour models can be made out of stacked boards of paperboard, cardboard, foam, or a soft wood like balsa or basswood. The thickness of the sheets should correspond to the vertical interval between contours at the desired map scale. To build a contour model by hand start by plotting a contour map at the desired map scale. Then spray the plotted paper map with a spray mount adhesive, and stick the plotted map onto a board. As an alternative to plotting and mounting, we can instead project the contour map. Cut out the lowest contour level with a precision knife such an x-acto. Glue this contour level onto the base or the level below. Repeat this process until all of the contour levels have been cut out and stacked together.

Hand sculpted models Hand sculpted sand and clay models are natural, intuitive, and even fun to make, but are imprecise and hard to read quantitatively (Fig. 3.2). When sculpting by hand we not only see, but also feel the 3-dimensional form, using our highly developed kinaesthetic intelligence to better understand the space. While tools can help us to cut sharp edges, shape certain forms, and smooth surfaces, sculpting with our hands alone lets us feel the shape of the topography and get an intuitive understanding of its form undistorted by perspective and depth perception. Because these models are continuous they could – if sculpted well – accurately represent the topography, but as they lack discrete intervals it is challenging to quantitively judge distances and heights. Because it is so natural and easy to make, but challenging to read quantitatively freeform hand modeling is better suited to ideation than detailed design or presentation.

While hand sculpted models have traditionally been made out of sand or clay, we recommend using polymer enriched sand. Clay holds form well, but can be sticky and hard to work with. Sand while easy to move does not hold a form well – sculpt too steep a slope and it will slump, the grains avalanching down. Polymeric sand is easy to work with and holds together well without sticking to our hands. It is not only easy to sculpt, but also easy to cast into precise forms. We will discuss methods for casting sand in digitally fabricated molds in Section 3.3. We use the product DeltaSand, which is also marketed as Kinetic Sand. It is made by mixing sand with a low-viscosity polymer and a cross-linking agent, resulting in a sand with the plasticity, moldability, adhesiveness, and longevity required for sculpting (Modell and Thuresson, 2009).

3.2 Digitally fabricated models

Complex physical models for Tangible Landscape can be digitally fabricated with CNC routing or 3D printing. Digitally fabricated models are rigid and precise – while they can not easily be sculpted, they are ideal presentation models, base models for object recognition, and molds for casting malleable models. CNC machines such as laser cutters, routers, and 3D printers require

digital instructions, a controller to decode the instructions, and a machining tool. Machining processes include subtraction with routers, addition with 3D printers, and deformation and molding with vacuum formers (Schodek et al., 2004, chap. 13). In this section we will explain how to prepare GIS data for digital fabrication and discuss various methods of digital fabrication. Section 3.4 describes workflows for creating models using these methods.

3.2.1 Digital models

Terrain and other continuous, 3D geographic data can be digitally represented as 3D points, 2.5D rasters, 3D rasters, contours, meshes, or surfaces. In GRASS GIS our workflows for landscape modeling and analysis are raster based. 3D printers and CNC routers, however, require vector data, typically meshes. After acquiring elevation data we import them into GRASS GIS and if necessary we interpolate a raster digital elevation model (DEM). This raster will later be used as the reference data for Tangible Landscape. Next, we export the raster as a point cloud and import it into a 3D modeling program to compute a surface, mesh, or solid from the point cloud. Finally, we generate a toolpath from the surface or mesh for the 3D printer or CNC router.

Digital elevation models The first step in modeling a landscape is to acquire data. Entire landscapes can be precisely 3D scanned as point clouds with airborne lidar. As pulses of light are emitted from the aircraft they hit and reflect back from vegetation, structures, and topography. When a pulse of light hits a tree part of the light is reflected and recorded as the first return, while the rest penetrates the outer canopy. The residual pulse, recorded as a series of returns, reflects off of leaves, branches, shrubs, and the ground. Lidar data can be used to compute very precise, high resolution models. By filtering lidar returns and classes we can extract points for the bare earth topography and interpolate a raster DEM or we can extract points with all of the structures and vegetation and interpolate a raster digital surface model (DSM). Lidar point clouds or ready to use raster DEMs are available for many locations and are well suited for building topographic models (see Appendix A.2 for links to lidar data and DEM repositories).

DEMs – raster maps of bare earth topography – and DSMs – raster maps of topography with structures and vegetation – are commonly used in GIS to represent and analyze terrain as they can easily be mathematically transformed. For example topographic parameters such as slope, aspect, and curvature can be computed from the partial derivatives of a function approximating a DEM or DSM (see Chapter 4).

Lidar, especially when filtered by classes or returns, can produce scattered, spatially heterogeneous data points that are challenging to accurately interpolate as terrain surfaces. Furthermore, other sources of elevation data such as digitized contours and surveying data can be clustered and have a

highly spatially heterogeneous distribution. The regularized spline with tension (RST) interpolation function, implemented as the *v.surf.rst* module in GRASS GIS, can be used to accurately model terrain surfaces from clustered and heterogeneous data such as lidar with some experimentation and tuning (see Section 4.1.2).

Contour data are challenging to interpolate because the data are spatially heterogenous – they are often very dense along the contour lines with large unsampled areas in regions with flat topography. If we acquire digital contour data we can either linearly interpolate between contours with *r.surf.contour* or thin the points along contours using *v.generalize*, convert them to a point representation and then interpolate them using a function like RST with the module *v.surf.rst* (The GRASS GIS Community, 2015).

Meshes Terrain can also be modeled in 3D vector representations such as polygon meshes or mathematically defined surfaces. In a mesh a vector network of polygons such as triangles or quadrilaterals forms a shape. Terrain meshes are typically triangulated irregular networks (TINs) that connect the data points from which they are computed. Triangulation was used to manually interpolate contours from surveyed spot elevations before the advent of computers. Now TINs are typically computed using the Delaunay triangulation algorithm which maximizes the smallest angle of every triangle in order to minimize the occurrence of very thin triangles. Since TINs connect the data points the input data is preserved. When meshes are low resolution, i.e. computed from small datasets, they reveal their polygonal structure with angled planes where there should be curved surfaces. When meshes are high resolution, i.e. computed from large datasets, they can be very detailed and represent highly complex forms. Even a high resolution mesh, however, is inaccurate when used to represent curved slopes as planar faces. Meshes are very useful for modeling structures and engineered topography with discontinuous slopes.

NURBS Non-uniform rational B-splines (NURBS) are parametric approximation curves and surfaces defined by a series of polynomials. The curves are basis splines defined by a knot vector, weighted control points, and the curve's order or degree (Piegl and Tiller, 1995). A NURBS surface is the tensor product of two NURBS curves (Martisek and Prochazkova, 2010). Since NURBS surfaces are mathematically defined they can precisely describe a continuous geometry.

Processed lidar data and rasters can be exported as point clouds from GIS and then imported into a 3D modeling program for meshing or surface generation. Toolpaths for CNC routing can be generated for a mesh or NURBS surface once it has been scaled. 3D printing, however, requires closed, solid volumes so the terrain surface should be offset or extruded to give it a thickness. Most 3D printers read the stereolithography (`.stl`) format, a mesh representation of a solid.

We typically use Rhinoceros for 3D modeling[1] with the RhinoTerrain plugin for terrain modeling[2] and the RhinoCAM plugin for CNC routing.[3] This proprietary 3D modeling program is useful for its interoperability – it can write and read a wide range of formats and can model NURBS, polygon meshes, solids, and point clouds. The RhinoTerrain plugin has modules for importing and exporting geographic data and efficiently computing large terrain meshes. While many computer aided manufacturing (CAM) programs require polygon meshes, the RhinoCAM plugin enables us to generate and simulate CNC toolpaths in Rhinoceros using NURBS, polygon meshes, or solids. The Rhino3DPRINT plugin can be used to prepare models for 3D printing.[4] Alternatives include a proprietary 3D modeling and animation program,[5] or MeshLab, an open source mesh processing program.[6]

3.2.2 Laser cutting

Laser cutting can be used to efficiently build precise contour models of moderate complexity (Fig. 3.3). Laser cutters are CNC machines that use a laser moving along a gantry in the x and y axes to make 2D cuts. To laser cut a contour model we separate each contour step onto different layers of a 2D vector drawing in a computer aided design (CAD) format (like `.dwg`, `.dxf`, or `.ai`). We cut each contour step out of a sheet of our material of choice and glue them together. We can also laser etch elevation values, patterns, and line work like road outlines and building footprints onto each layer. While it is easy to precisely cut complex topographies it is still challenging to construct them as we have to lay and glue each piece by hand.

The materials determine the cost. As this is a subtractive method a lot of material is wasted. Contour models cut from thick, low-cost materials like cardboard are relatively inexpensive, but have large contour steps and thus low resolution. Models cut from thin, low-cost materials like chipboard allow for small contour steps at moderate cost, but tend to warp with time especially if too much glue has been applied. When aesthetics and archivability are important museum board or acrylic can be cut to create presentation-quality laser cut contour models. Paper-based media such as chipboard or museum board are not suitable for casting polymeric sand as the polymer will stick and soak into them.

---

[1] Rhinoceros: `http://www.rhino3d.com/`

[2] RhinoTerrain: `http://www.rhinoterrain.com/`

[3] RhinoCAM: `http://mecsoft.com/rhinocam-software/`

[4] Rhino3DPRINT: `http://mecsoft.com/rhino3dprint/`

[5] Maya: `http://www.autodesk.com/products/maya/overview`

[6] MeshLab: `http://meshlab.sourceforge.net/`

Fig. 3.3: A laser cut chipboard contour model with basswood buildings fabricated by David Koontz and Faustine Pastor

3.2.3 CNC routing

CNC routing or milling is a subtractive fabrication process that can be used to precisely manufacture contour models and surface models of great complexity (Fig. 3.4). This type of digital fabrication is a precise, accurate, inexpensive, and scalable way to build terrain models. 3-axis CNC routers and milling machines move a spindle with a machining bit along the x, y, and z axes in a programmed path to carve a shape out of a block of material (Fig. 3.5). Because 3-axis routers can only carve vertically they produce 2.5D surface models on a solid base, rather than full 3D volumetric models. It is possible, however, to manufacture fully 3D volumes with 3-axis routers; after each cut one can reorient the model and then route the sides or bottom of the model (Schodek et al., 2004, chap. 14).

Fig. 3.4: A CNC routed terrain model of the Jockey's Ridge dune complex on the Outer Banks of North Carolina

To CNC route a terrain model we first prepare a solid block of our material. We can use materials such as medium to high density foam, wax, and wood products. We typically use medium density fiberboard (MDF) or baltic birch, engineered wood products with high strength, good dimensional stability, and a fine, uniform grain with minimal voids. To prepare MDF we cut the sheet into tiles that match the desired extent of our scale model, spread wood glue

Fig. 3.5: A 3-axis CNC router carving a terrain model

evenly over each tile and stack the tiles together, and firmly clamp the stack of tiles together until the glue sets (Fig. 3.6).

(a) Pour glue onto a layer of MDF

(b) Spread glue evenly over a layer of MDF

(c) Stack layers of MDF

(d) Clamp the stack of MDF

(e) Add a weight on top of the stack of MDF

(f) Wait for the glue to dry

Fig. 3.6: Preparing MDF for CNC routing

In a CAD program we prepare a digital model of the geometry we wish to carve. Then we generate a toolpath for this digital model in a CAM program. To make a contour model we use closed, 3D contour curves as our data and then carve with a contour cut. Contour cutting can also be used to carve buildings. To carve a surface model from a mesh or NURBS surface we use parallel cuts. For a surface model we typically start with a horizontal rough cut with a 0.5 inch diameter, carbide bit to remove the bulk of the excess material (Table 3.1). Then we use a parallel finish cut with a 0.25 inch diameter, carbide, ball-nose bit to carve the the terrain as a continuous surface (Table 3.2). If we need a more refined presentation-quality model we then make two more parallel finish cuts in alternative cutting directions with a 0.125 inch diameter, carbide, ball-nose bit to smooth the surface and capture more detail. Finally a contour cut along the border can be used to neatly trim and remove the model from the base material. The toolpath is written as a sequence of instructions in a CNC programming language often as a `.nc` file in G-Code. The CNC machine's controller reads this code and drives the tool along the toolpath carving our model. We have a streamlined workflow using Rhinoceros with the RhinoCAM plugin for both 3D modeling and the generation, simulation, and visualization of toolpaths. We do our CAD and CAM in the same environment – Rhinoceros – so that we can easily edit our models and work with both meshes and NURBS.

Once the model has been routed we may want to finish it by sanding, priming, and painting the model. Sanding the model, a prerequisite for priming and painting, with a fine grit sandpaper produces a smoother surface. After routing and sanding we clean the model with an air hose. Applying several coats of magnetic primer to our model weakly magnetizes it so that magnetized markers stick to the slopes (Fig. 3.7). A magnetized model is useful for Tangible Landscape applications relying on object recognition rather than sculpting. Finally we may want to spray paint our model white to enhance the brightness of projected imagery.

Table 3.1: CNC settings: horizontal rough cut with MDF

Bit ∅	Speed	Plunge	Approach	Engage	Cut	Retract	Depart.	Step down	Stepover
0.5 inch	16000	65	135	205	275	800	800	0.32 inch	48%
0.25 inch	16000	50	75	125	175	800	800	0.25 inch	40%

3.2.4 3D printing

With 3D printing, a type of solid freeform fabrication, we can make a complex, 3D volume in a single run. While the models can be precise, accurate, and highly complex, they are also expensive and small as 3D printers have

Fig. 3.7: A marker with a magnetic base sticking to CNC-routed terrain model primed with magnetic paint

(a) (b)

Fig. 3.8: Modes of interaction using markers: (a) rasterizing an area and (b) drawing a polygon.

Table 3.2: CNC settings: parallel finish cut with MDF

Bit ∅	Speed	Plunge	Approach	Engage	Cut	Retract	Departure	Stepover
0.5 inch	16000	65	160	230	325	800	800	20%
0.25 inch	16000	50	100	150	225	800	800	20%

restrictively small build areas and the materials are relatively expensive. 3D printing is an ideal process for fabricating small, complex models such as buildings or small, high quality presentation models for use with Tangible Landscape (Fig. 3.9a).

While CNC milling and routing are subtractive processes, 3D printing is an additive process in which a model is built up layer by layer. A digital, 3D, solid model is divided into a stack of horizontal layers or slices and a toolpath is computed for each slice. The physical model is then built slice by slice by depositing or hardening material along the toolpath. By dividing the model into cross sections, each as thin as the technology allows, a complex volume can be formed. There are a variety of different 3D printing processes including stereolithography, selective laser sintering, fused deposition modeling, and 3D ink-jet printing each with tradeoffs in build speed, quality, strength, cost, resolution, color, and material (Schodek et al., 2004, chap. 14).

3.3 Molding and casting

To sculpt with Tangible Landscape we need a malleable model made of a soft, deformable medium. A model made of polymer enriched sand is easy to sculpt, holds its form well, and can be cast into precise forms. CNC routed and 3D printed models can be used as molds for casting polymeric sand into malleable models. The mold should be the inverse of the surface that will be cast. Cast models can precisely represent complex forms that are too challenging to model by hand and can easily be recast.

To cast a terrain model either CNC route the inverse of the terrain (Fig. 3.10) or 3D print the terrain as a volume extruded with enough thickness to survive the casting process (Fig. 3.9). We press the mold into the polymeric sand to cast the form. We have to check the cast and repeat the process if necessary – sand may need to be added or removed in places to get a good cast. Clamps can be used to get strong, even pressure on a CNC routed mold when casting.

Thermoforming or vacuuming forming can be used to quickly make lightweight copies or imprints of a CNC routed terrain model in a thermoplastic. Since they are lightweight thermoformed models are very portable and are ideal for casting polymeric sand while traveling. To thermoform a terrain model we heat a thermoplastic sheet in a vacuum former until it becomes soft. Use the vacuum to pull the hot plastic over a mold deforming the plastic into the desired shape. Once the plastic cools into the cast shape we release the vacuum. We may need to drill small holes through our mold to make a complete vacuum (Schodek et al., 2004, chap. 14).

(a) 3D prints of the terrain and canopy

(b) Casting sand with 3D prints

(c) The canopy cast in sand

Fig. 3.9: Casting polymeric sand from 3D printed molds

(a) Pour polymeric sand onto the base

(b) Stack the inverse model on top of the sand

(c) Apply pressure

(d) Check the cast and remold if necessary

(e) Trim the excess sand

(f) Remove the inverse model

Fig. 3.10: Cast polymeric sand from CNC routed molds

Fig. 3.11: A thermoformed polystyrene model of part of Lake Raleigh Woods cast over a CNC routed mold.

3.4 Workflows

This section describes workflows for creating physical models for Tangible Landscape. Our examples use lidar point cloud data for Lake Raleigh Woods on North Carolina State University's (NCSU) Centennial Campus acquired during a 2013 lidar survey by Wake County. The data are provided as a single tile in the standard LAS format (`tile_0793_015_spm.las`). We processed this data within a GRASS GIS data set `nc_spm_tl`. All examples in the following subsections assume running GRASS GIS with this data set; see Appendix A.3 for information on how to download the data, start GRASS GIS with this location data set, and perform basic display and other operations.

(a) (b)

Fig. 3.12: 3D renderings of terrain models derived from lidar tile 0793_015 using (a) GRASS GIS and (b) Rhinoceros with the RhinoTerrain plugin (see Section 3.4.4)

3.4.1 Selecting a 3D model scale

The scale of the physical 3D model depends on the extent of the data, the desired model size, and the desired scale. The scale may also depend on the material of the model, the scanning technology, and the resolution of the data. Here we assume that we know the spatial extent of our data and want our model to be approximately half a meter long on each side. If we do not already know the extent of our data we can determine the values once the data has been imported into GRASS GIS for example using *v.info* or *r.info*.

We can perform this computation in the Python interactive console. First, we enter the spatial extent of the data and approximate the size of the model (in meters):

```
n = 224028.45
s = 223266.45
e = 639319.28
w = 638557.28
desired_model_x = 0.50
```

Then we compute the scale using the equation:

$$s = \frac{d_m}{d_r} \tag{3.1}$$

where s is the scale of the model, d_m is the distance measured on the model, and d_r is the real-world distance. For convenience we can also compute the scale number using the equation:

$$s_n = \frac{1}{s} \tag{3.2}$$

where s_n is the scale number of the model and s is the scale of the model.

We then type the following in the Python console:

```
scale = desired_model_x / (n - s)
scale_number = 1. / scale
```

The scale is now 1 : 1524 and the scale number is 1524. Since we want the scale to be a round number and do not want to add or remove data we need to modify the scale and reverse the computation in the previous step so that we can compute the new size of the model at the selected scale. We select a scale of 1 : 1500 (a round value close to the previously computed scale). In Python we then write:

```
scale_number = 1500
scale = 1. / scale_number
model_x = (n - s) * scale
```

The resulting model size is 50.8 cm.

So far we have only used the spatial extent along the north-south direction. We skipped the east-west direction because the extent of both the data and the model were squares. In a region with an elongated rectangular shape we might need to determine a suitable scale for each of the horizontal directions and then decide on a compromise.

Next we determine the vertical size of the model and its vertical exaggeration. First, we enter the minimum and maximum elevation in the dataset (called top and bottom in GRASS GIS) and we also include the desired height of the model (in meters). In our example we set the desired height of the model to 0.04 m:

```
t = 109.33
b = 76.54
desired_model_h = 0.04
```

Then we compute the model height without vertical exaggeration and the exaggeration based on the desired model height using the equation:

$$e = \frac{h_e}{h} \tag{3.3}$$

where e is the exaggeration, h_e is the exaggerated height (in our case the desired height of the model) and h is the height of the model according to the scale in case the model would not be exaggerated.

Since we already know the scale of the model from the previous computations we can now compute the exaggeration and model height without vertical exaggeration:

```
model_h = scale * (t - b)
exaggeration = desired_model_h / model_h
```

With the given the inputs the exaggeration is now approximately 1.8. Since we want a round number we set the vertical exaggeration to 2 and compute the actual height of the model using Eq. 3.3:

```
exaggeration = 2
actual_model_h = exaggeration * model_h
```

With the exaggeration set to 2 the model height (or more precisely the maximum height difference of the top surface of the model) will be approximately 4.4 cm. The base of the model will increase the actual model height. Depending on how it is constructed the base may or may not be an integral part of the model. If we set the exaggeration to 1.5 the model height difference would be approximately 3.3 cm, which might be too shallow. At a different scale, however, an exaggeration of 1.5 might be the right choice.

We have used the same scale in both the x and y horizontal directions (1 : 1500). However, in the vertical direction (z) we exaggerated the scale by a factor of 2. As a result the vertical scale (1 : 750) is different than horizontal scale (1 : 1500). Depending on the audience it may be advantageous to note either the horizontal and vertical scales or the scale and exaggeration. When the scale and exaggeration are known the vertical scale can be computed using the equation:

$$s_v = se \tag{3.4}$$

where s_v is the vertical scale, s is the horizontal scale and e is the exaggeration. Once we have computed the size of our model we can build the model using any of the methods described in the following sections.

3.4.2 Sculpting a malleable model from data

A simple way to create a malleable 3D model is to sculpt the landscape in polymer-enriched sand using projected contours derived from DEM as a guide. In this workflow we used a point cloud of elevation data in the LAS format (`tile_0793_015_spm.las`) and worked in the GRASS GIS location `nc_spm_tl`. To sculpt a malleable model from lidar data perform the following steps:

Import the lidar points classified as ground (standard class 2) into GRASS GIS using the module *v.in.lidar*:

```
v.in.lidar -t input=tile_0793_015_spm.las output=ground \
    class_filter=2
```

Set the region to the spatial extent of the imported tile and resolution to 1 m. Then interpolate the DEM from the processed lidar data:

```
g.region vector=ground res=1
v.surf.rst -t input=ground elevation=dem tension=100 npmin=250 \
    dmin=2
```

Check the results, adjust the parameters, and rerun with the overwrite flag if necessary. Then compute the 1 m interval contours from the DEM:

```
r.contour input=dem output=contours_1m step=1
```

We can now sculpt the terrain in polymer-enriched sand using the projected DEM and contours as a guide (Fig. 3.2). Once we have made at least a rough approximation of the form, we can use DEM differencing to critique and refine the modeled form (see Section 4.2.2).

3.4.3 CNC routing a topographic model from contour data

Import the contours provided in `.dxf` file into Rhinoceros:

```
_Import contours.dxf _Enter
```

Zoom to the extent of all viewports:

```
Zoom All Extents
```

Select the contours and interpolate a terrain mesh:

```
_RtMeshTerrainCreate _Accept _Enter
```

Scale the model uniformly in the x, y, z direction to our chosen map scale by selecting the mesh, setting the origin to 0,0,0 and setting the scale factor:

```
_Scale 0,0,0 1/450 _Enter % 1/1500 ft
```

Optionally scale the model in the z-direction to exaggerate the relief by selecting the mesh, setting the origin to 0,0,0, setting the scale factor to 2, and setting the scale direction by drawing a line in the z-axis with ortho mode, using the gumball, or entering the coordinates 0,0,0 and 0,0,1:

```
_Scale1D 0,0,0 2 0,0,0 0,0,1 _Enter
```

Use RhinoCAM to generate the toolpath. Then CNC route the model.

3.4.4 CNC routing topographic and surface models from lidar data

Import the lidar data classified as ground (standard class 2) into GRASS GIS:

```
v.in.lidar -t input=tile_0793_015_spm.las output=ground \
    class_filter=2
```

Import the first return lidar data into GRASS GIS filtering unnecessary points:

```
v.in.lidar -t input=tile_0793_015_spm.las output=surface \
    return_filter=first class_filter=1,2,3,4,5,6,9
```

Set the region and resolution and interpolate the DEM from the ground points:

```
g.region vector=ground res=1
v.surf.rst -t input=ground elevation=dem tension=100 npmin=250 \
    dmin=2
```

Check the results, adjust the parameters, and rerun with the overwrite flag if necessary. With the region already set in the previous step, interpolate the DSM from the surface points:

```
v.surf.rst -t input=surface elevation=dsm tension=200 \
    smooth=0.5 npmin=120 dmin=2
```

Check the results, adjust the parameters, and rerun with the overwrite flag if necessary. Export the DEM to a space delimited text file with x, y, and z values:

```
r.out.xyz input=dem output=dem.xyz separator=space
```

Export the DSM to a space delimited text file with x, y, and z values:

```
r.out.xyz input=dsm output=dsm.xyz separator=space
```

Import the DEM text file into Rhinoceros as a point cloud. In the import options, set delimiter to 'space' and check 'create point cloud':

```
_Import dem.xyz
```

Import the DSM text file as a point cloud. In the import options, set delimiter to 'space' and check 'create point cloud':

```
_Import dsm.xyz
```

Zoom to the extent of all viewports:

```
Zoom All Extents
```

Select the DEM point cloud, interpolate a terrain mesh, and delete or hide the point cloud:

```
_RtMeshTerrainCreate _Accept _Enter
```

Select the DSM point cloud and interpolate a terrain mesh. Delete or hide the point cloud:

```
_RtMeshTerrainCreate _Accept _Enter
```

Scale both the DEM and DSM meshes uniformly in the x, y, z direction to our chosen map scale by selecting the meshes, setting the origin to 0,0,0 and setting the scale factor:

```
_Scale 0,0,0 1/1500 _Enter
```

Optionally scale both the DEM and DSM meshes in the z-direction to exaggerate the relief by selecting the meshes, setting the origin to 0,0,0, setting the scale factor to 2, and setting the scale direction by drawing a line in the z-axis with ortho mode, using the gumball, or entering the coordinates 0,0,0 and 0,0,1:

```
_Scale1D 0,0,0 2 0,0,0 0,0,1 _Enter
```

Optionally, create a vector curve delineating the border of the DEM or DSM:

```
_RtMeshFindBorder _Enter
```

Use RhinoCAM to generate the toolpaths. CNC route the models.

3.4.5 3D printing topographic and surface models from lidar data

Import the lidar data classified as ground (standard class 2) into GRASS GIS:

```
v.in.lidar -t input=tile_0793_015_spm.las output=ground \
    class_filter=2
```

Set the region to the given tile and set a relatively low resolution of 10 m. Interpolate the DEM from the ground points:

```
g.region vector=ground res=10
v.surf.rst -t input=ground elevation=dem tension=100 npmin=250
```

Export the DEM to a space delimited text file with x, y, z values:

```
r.out.xyz input=dem output=dem.xyz separator=space
```

Import the DEM text file into Rhinoceros as a point cloud. In the import options, set delimiter to 'space' and check 'create point cloud':

```
_Import dem.xyz _Enter
```

Zoom to the extent of all viewports:

```
Zoom All Extents _Enter
```

Select the DEM point cloud and interpolate a terrain mesh. Delete or hide the point cloud:

```
_RtMeshTerrainCreate _Accept _Enter
```

Scale the mesh uniformly in the x, y, z direction to our chosen map scale by selecting the mesh, setting the origin to 0,0,0 and setting the scale factor:

```
_Scale 0,0,0 1/3000 _Enter
```

Optionally scale the mesh in the z-direction to exaggerate the relief by selecting the mesh, setting the origin to 0,0,0, setting the scale factor to 2, and setting the scale direction by drawing a line in the z-axis with ortho mode, using the gumball, or entering the coordinates 0,0,0 and 0,0,1:

```
_Scale1D 0,0,0 2 0,0,0 0,0,1 _Enter
```

Create a vector curve delineating the border of the DEM:

```
_RtMeshFindBorder _Enter
```

Prepare the mesh for 3D printing. Select the border curve as the boundary and the DEM mesh as the mesh. Set offset to relative or absolute. Set a base height:

```
_RtMesh3dPrintPrepare
```

Export the selected mesh as a stereolithography file (`.stl`) and send it to the 3D printer:

```
_SaveAs
```

3.4.6 Casting a malleable topographic model with a CNC routed mold derived from lidar data

Import the lidar data classified as ground (standard class 2) into GRASS GIS:

```
v.in.lidar -t input=tile_0793_015_spm.las output=ground \
    class_filter=2
```

Set the region and resolution, then interpolate the DEM from the ground points:

```
g.region vector=ground res=1
v.surf.rst -t input=ground elevation=dem tension=100 npmin=250 \
    dmin=2
```

Check the results, adjust the parameters, and rerun with the overwrite flag if necessary. Export the DEM to a space delimited text file with x, y, z values:

```
r.out.xyz input=dem output=dem.xyz separator=space
```

Import the DEM text file into Rhinoceros as a point cloud. In the import options, set delimiter to 'space' and check 'create point cloud':

```
_Import dem.xyz _Enter
```

Zoom to the extent of all viewports:

```
Zoom All Extents _Enter
```

Select the DEM point cloud and interpolate a terrain mesh. Delete or hide the point cloud:

```
_RtMeshTerrainCreate _Accept _Enter
```

Optionally, create a vector curve delineating the border of the DEM:

```
_RtMeshFindBorder _Enter
```

Invert the DEM by rotating it 180 degrees on the z axis. Set the 'first reference point' by drawing a vertical line with ortho mode on and then set the 'second reference point' to '180' or by or entering the coordinates 0,0,0 and 0,0,1 and then 180 in the command line:

```
_Rotate 0,0,0 0,0,1 180 _Enter
```

Scale the mesh uniformly in the x, y, z direction to our chosen map scale by selecting the mesh, setting the origin to 0,0,0 and setting the scale factor:

```
_Scale 0,0,0 1/1500 _Enter
```

Optionally scale the mesh in the z-direction to exaggerate the relief by selecting the mesh, setting the origin to 0,0,0, setting the scale factor to 2, and setting the scale direction by drawing a line in the z-axis with ortho mode, using the gumball, or entering the coordinates 0,0,0 and 0,0,1:

```
_Scale1D 0,0,0 2 0,0,0 0,0,1 _Enter
```

Optionally, create a vector curve delineating the border of the DEM or DSM:

```
_RtMeshFindBorder _Enter
```

Use RhinoCAM to generate the toolpath, CNC route the inverted terrain model, and use this routed model as a mold to cast polymeric sand into a solid, malleable model of the topography.

References

Martisek, D. and Prochazkova, J. (2010). Relation between algebraic and geometric view on nurbs tensor product surfaces. *Applications of Mathematics*, 55(5):419–430. Copyright - Institute of Mathematics of the Academy of Sciences of the Czech Republic, Praha, Czech Republic 2010; Last updated - 2014-08-22.

Modell, J. and Thuresson, S. (2009). Material composition and method for its manufacture. EP Patent App. EP20,070,794,114.

Piegl, L. and Tiller, W. (1995). *The NURBS book*. Springer, New York.

Schodek, D., Bechthold, M., Griggs, K., Kao, K. M., and Steinberg, M. (2004). *Digital Design and Manufacturing*. John Wiley & Sons, Inc., Hoboken, New Jersey.

The GRASS GIS Community. Contour lines to DEM [online]. (2015) [Accessed 2015-05-28]. http://grasswiki.osgeo.org/wiki/Contour_lines_to_DEM.

Chapter 4
Basic landscape analysis

Tangible Landscape allows us to explore the spatial patterns of topographic parameters and their relation to basic surface geometry. We can easily sculpt a physical model of a landscape with our hands changing its topography. With Tangible Landscape we can analyze the topography of a landscape model and how it changes by continually 3D scanning the model and computing DEMs from the scanned point clouds using binning or interpolation. By computing basic topographic parameters, morphometric units, and DEM differencing we can map changes in elevation, slope, and landform as the model is modified. These maps are then projected over the physical model of the landscape so that we have near real-time feedback and can understand the impact of the changes we are making as we make them.

4.1 Processing and analyzing the scanned DEM

To start working with Tangible Landscape, the setup is first calibrated using a flat surface, typically a table, to minimize the scanning distortion (see Section 2.2.4). After successful calibration the model can be scanned continuously. With each scan the edges of the model are detected and georeferenced and the point cloud is converted into a DEM via binning or interpolation. Standard GRASS GIS modules are then used to compute maps of topographic parameters or landforms that are then projected over the model. This chapter explains how the DEM is derived and demonstrates the results in a case study.

4.1.1 Creating DEM from point cloud

There are two ways to derive a raster DEM from a set of points in Tangible Landscape:

Binning A cell value is assigned based on the univariate statistics of the points that fall inside the particular cell. When constructing a DEM the mean of the z coordinates is typically used. Binning is fast, but creates a rough surface so it is only useful for certain types of analyses.

Interpolation Cell values are estimated from available point data using a selected mathematical or statistical interpolation technique.

4.1.2 Interpolation with the RST function

The point clouds produced by the scanning process are usually noisy. The regularized spline with tension (RST) function, implemented as the *v.surf.rst* module in GRASS GIS, can be used to interpolate smooth DEMs from noisy point data and simultaneously compute topographic parameters such as slope and curvature. RST approximates a surface from data points by minimizing a smoothness seminorm and the deviations between the given points and the resulting surface (Mitasova et al., 2005). The RST smoothness seminorm includes derivatives of all orders with their weights decreasing with the increasing derivative order leading to the function:

$$z(\mathbf{r}) = a_1 + \sum_{j=1}^{N} \lambda_j R(\varrho_j) \tag{4.1}$$

$$R(\varrho_j) = -[E_1(\varrho_j) + \ln(\varrho_j) + C_E] \tag{4.2}$$

where $z(\mathbf{r})$ is the elevation at a point $\mathbf{r} = (x, y)$, a_1 is a trend, λ_j are coefficients, N is a number of given points, $R(\varrho_j)$ is a radial basis function, $\varrho_j = (\varphi r_j/2)^2$, φ is a generalized tension parameter, $r_j = |\mathbf{r} - \mathbf{r}_j|$ is a distance, $C_E = 0.577215$ is the Euler constant, and $E_1(\varrho_j)$ is the exponential integral function (Abramowitz and Stegun, 1965; Mitasova and Mitas, 1993). The coefficients a_1, $\{\lambda_j\}$ are obtained by solving the system of linear equations:

$$\sum_{j=1}^{N} \lambda_j = 0 \tag{4.3}$$

$$a_1 + \sum_{j=1}^{N} \lambda_j \left[R(\varrho_j) + \delta \frac{w_0}{w_j} \right] = z(\mathbf{r}_i), \qquad i = 1, \ldots, N \tag{4.4}$$

where w_0/w_j are positive weighting factors representing a smoothing parameter at each given point $\mathbf{r_j} = (x_j, y_j)$.

Tension influences the detail of the surface and smoothing influences the deviations between the given points and the resulting surface. A higher smoothing value can be used to reduce the noise in the data. Because this is a global operation acting on the entire dataset it can be processing intensive for large datasets so the *v.surf.rst* implementation uses a quadtree segmentation

algorithm with smooth overlaps to separately process overlapping neighborhoods of points. A quadtree is a tree data structure that recursively partitions processing regions into smaller processing regions by dividing them into quadrants. Depending on the distribution of the input points in relation to the surface complexity, interpolation with *v.surf.rst* may require experimental, iterative tuning of the parameters. By adjusting the parameters which control the RST function properties and distribution of points used for interpolation, this function can be used to accurately model smooth or rough topography (Mitasova and Mitas, 1993; Mitasova et al., 2005).

4.1.3 Analyzing the DEM

The geometry of an elevation surface at any point can be described by topographic parameters – slope, aspect, and several types of curvatures (Mitasova et al., 2005; Olaya, 2009). Elevation surface can be partitioned into units with specific geometric properties; one approach to defining these units is geomorphons.

Basic topographic parameters The steepest slope angle β in degrees or percent and the aspect angle α in degrees are the most commonly used topographic parameters. They represent the magnitude and direction of the surface gradient vector $\nabla z = (f_x, f_y)$ (its direction is upslope) and are computed as follows:

$$\beta = \arctan\sqrt{f_x^2 + f_y^2} \qquad \beta[\%] = 100\sqrt{f_x^2 + f_y^2} \tag{4.5}$$

$$\alpha = \arctan -\frac{f_y}{f_x} \tag{4.6}$$

where $f_x = \partial z/\partial x$ and $f_y = \partial z/\partial y$ are the first order partial derivatives of elevation surface function $z = f(x, y)$. The slope values range from 0° to 90° and the aspect values range from 0° to 360°. Therefore, computing the correct angle for aspect requires the evaluation of all possible combinations of negative, positive and zero values of f_x, f_y in relation to the selected direction of 0° (usually east or north).

For applications in geosciences the curvature in the gradient direction (profile curvature) is important because it reflects the change in the slope angle and thus controls the change of the velocity of mass flowing downwards along the slope curve. The equation for the profile curvature is

$$\kappa_s = \frac{f_{xx}f_x^2 + 2f_{xy}f_xf_y + f_{yy}f_y^2}{p\sqrt{q^3}} \tag{4.7}$$

where κ_s is the profile curvature in m^{-1} and f_{xx}, f_{xy}, f_{yy} are the second order partial derivatives of the elevation surface function $z = f(x, y)$, $p = f_x^2 + f_y^2$

and $q = p+1$. The curvature in a direction perpendicular to the gradient (tangential curvature) reflects the change in the aspect angle and influences the divergence/convergence of water flow. The equation for tangential curvature at a given point is

$$\kappa_t = \frac{f_{xx} f_y^2 - 2 f_{xy} f_x f_y + f_{yy} f_x^2}{p\sqrt{q}} \tag{4.8}$$

where κ_t is the tangential curvature in m^{-1}. Partial derivatives can be computed from a suitable approximation function $z = f(x, y)$ such as local second order polynomial or certain types of splines.

Partial derivatives from a raster DEM We can use a second order polynomial approximation of a surface defined by a given grid point and its 3×3 neighborhood to estimate the topographic parameters (Horn, 1981):

$$z(x, y) = a_0 + a_1 x + a_2 y + a_3 xy + a_4 x^2 + a_5 y^2 \tag{4.9}$$

By fitting this polynomial to the 9 grid points (the given point $z_{i,j}$ and its 3×3 neighborhood as shown below in Fig. 4.1) using weighted least squares we can derive the coefficients of this polynomial as well as its partial derivatives ($f_x = a_1$, $f_y = a_2$, $f_{xx} = 2a_4$, $f_{yy} = 2a_5$, $f_{xy} = a_3$):

$z_{i-1,j+1}$	$z_{i,j+1}$	$z_{i+1,j+1}$
$z_{i-1,j}$	$z_{i,j}$	$z_{i+1,j}$
$z_{i-1,j-1}$	$z_{i,j-1}$	$z_{i+1,j-1}$

Fig. 4.1: The elevation values of a grid cell and its 3×3 neighborhood

$$f_x = \frac{(z_{i-1,j-1} - z_{i+1,j-1}) + 2(z_{i-1,j} - z_{i+1,j}) + (z_{i-1,j+1} - z_{i+1,j+1})}{8\Delta x} \tag{4.10}$$

$$f_y = \frac{(z_{i-1,j-1} - z_{i-1,j+1}) + 2(z_{i,j-1} - z_{i,j+1}) + (z_{i+1,j-1} - z_{i+1,j+1})}{8\Delta y} \tag{4.11}$$

where Δx and Δy is the resolution (grid spacing) in the east-west and north-south directions respectively. To compute the second order partial derivatives

we first denote $D(i,\delta) = z_{i,j+1} + z_{i,j-1} - 2z_{i,j}$ and $D(\delta,j) = z_{i+1,j} + z_{i-1,j} - 2z_{i,j}$. Then we can write:

$$f_{xx} = \frac{D(\delta,j+1) + (4z_{i-1,j} + 4z_{i+1,j} - 8z_{i,j}) + D(\delta,j-1)}{6(\Delta x)^2} \tag{4.12}$$

$$f_{yy} = \frac{D(i-1,\delta) + 4(z_{i,j+1} + z_{i,j-1} - 2z_{i,j}) + D(i+1,\delta)}{6(\Delta y)^2} \tag{4.13}$$

$$f_{xy} = \frac{(z_{i-1,j-1} - z_{i+1,j-1}) - (z_{i-1,j+1} - z_{i+1,j+1})}{4\Delta x \Delta y} \tag{4.14}$$

where $z_{i,j}$ is the elevation value at row j and column i, Δx is the east-west grid spacing, and Δy is the north-south grid spacing (resolution).

(a) (b)

Fig. 4.2: Slope (a) and aspect (b) maps in degrees draped over a DEM

(a) (b)

Fig. 4.3: Profile (a) and tangential (b) curvature maps draped over a DEM

Landforms Landforms can be identified automatically by fitting a quadratic function to the elevation values in a given neighborhood or 'moving window'

using least squares. This method for identifying landforms using differential geometry is implemented in GRASS GIS as a module *r.param.scale*. While real-world landforms are scale-dependent and may be nested, this method can only identify landforms at a single scale based on the size of the moving window.

Geomorphons – geomorphologic phonotypes – is a novel method for identifying landforms using pattern recognition developed by Jasiewicz and Stepinski (2013) and implemented in GRASS GIS as the add-on module *r.geomorphon*. A geomorphon is an abstract unit of terrain described by the local ternary pattern rather than by relief. The relationship between a cell and its neighborhood is described using an 8-tuple pattern of lower (-), equal (0), higher (+) elevation values (Fig. 4.5a). Neighborhoods are based on visibility rather than the predefined dimensions of a moving window. Geomorphons uses the line-of-sight principle to dynamically determine the optimal size of the search radius for a neighborhood based on openness of the terrain (see Fig. 4.5b). Thus, unlike methods based on differential geometry, the geomorphons approach is able to efficiently classify landforms across a range of spatial scales (Jasiewicz and Stepinski, 2013).

Fig. 4.4: Landforms identified by geomorphons (see *r.geomorphon* manual page for explanation of landform types).

4.2 Case study: topographic analysis of graded landscape

In this section we demonstrate how changes in elevation impact topographic parameters and landforms in a small study area on NCSU's Centennial Campus.

(a) (b)

Fig. 4.5: Geomorphons: local ternary patterns and line-of-sights concepts applied to landform identification. The relationship between a cell and its neighbors in terms of relative elevation (higher than, equal to, or lower than) is expressed as a ternary pattern (a). Ternary patterns are derived using the line-of-sight method and comparing the zenith angle Φ and nadir angle Ψ (b).

4.2.1 Site description and model

Our 56 ha case study area is located on NCSU's Centennial Campus on the south-west side of Lake Raleigh (Fig. 4.6). The topography is largely natural, but there are some anthropogenic features, such as informal trails and the Chancellor's house with its access road, parking, and retention pond. There is rapid development on the campus; several new buildings are currently under construction. At the same time there is a desire to preserve part of the campus as a park and a natural teaching area. In this case study we developed part of the natural area as student housing and constructed an access road. Since significant grading would be required to develop the site given the steep slopes we extensively modified the topography and mapped the change in elevation (cut and fill), slope, aspect, curvature, and landform.

We CNC routed the base and mold for the model from MDF based on a DEM derived from a 2013 lidar survey. We used the base and the mold to cast a sculptable sand model. Each side of the 1 : 1500 scale model is 50 cm and represents 750 m. The model is 2 times vertically exaggerated and has 44 mm of relief which represent 33 m of elevation change.

We sculpted the sand model to grade a road and building sites. With Tangible Landscape we were able to analyze these modifications and understand how we had changed the slope, aspect, shape, and volume of the landscape.

4.2.2 Basic workflow with DEM differencing

To explore the impact of the new development we graded building sites and a road for the future neighborhood. We start with the original molded sand

Fig. 4.6: Lake Raleigh Woods with our study site highlighted

model, scan it and save the scan under different name so that it is not overwritten by the next scan. Then we compute and display 1-meter contours:

```
g.rename raster=scan,scan_before
g.region raster=scan_before
r.contours input=scan_before output=contours_1m step=1
```

Fig. 4.7a shows the terrain conditions before grading. While sculpting the model, we can pause the continuous scanning or keep it running to get ongoing feedback on our design as we progress.

We can visualize the magnitude and extent of our changes by differencing the scanned elevation before and after the change and then assigning a diverging color scheme to helps us distinguish between cut and fill (Fig. 4.7d):

```
g.rename raster=scan,scan_after
r.mapcalc "diff = scan_after - scan_before"
r.colors -n map=diff color=differences
```

Because the scans are noisy, they do not align precisely in vertical direction. When computing difference between scans, any vertical shift is undesirable and should be minimized. The following workflow uses linear regression to match one scan to the other on the assumption that there has only been a moderate change in elevation. The linear regression of the form $y = a + bx$ estimates the vertical shift a and scale coefficient b (which should be close to 1) for adjusting the scan. Here we define a function in Python for adjusting the scan to match the other; we will refer to this function throughout the book:

Fig. 4.7: Basic workflow: (a) The scanned elevation with contours, a red curve indicates the planned road; (b) modifying the terrain; (c) the finished design; (d) and the difference in meters between the elevation before and after grading, red representing soil removal (cut) and blue representing soil deposition (fill).

```
def adjust_scan(scan_before, scan_after, scan_adjusted):
   coeff = gscript.parse_command('r.regression.line',
       mapx=scan_after, mapy=scan_before, flags='g')
   gscript.mapcalc("{scan_adjusted} = {a} + {b} *
       {scan_after}".format(scan_adjusted=scan_adjusted,
       a=coeff['a'], b=coeff['b'], scan_after=scan_after))
```

Here we use the function for adjusting scans to compute the difference:

```
adjust_scan('scan_before', 'scan_after', 'scan_adjusted')
gscript.mapcalc("diff = scan_adjusted - scan_before")
gscript.run_command('r.colors', flags='n', map='diff',
    color='differences')
```

4.2.3 The impact of model changes on topographic parameters

In our case study the new neighborhood is planned on an east-facing hillside with a gentle slope of about 5 degrees (Fig. 4.8). The slope values are steep

on two sides of the hillside where it changes into parallel valleys; the profile curvature increases at the edge of the valleys (Fig. 4.9). The profile curvature on the hillside varies between positive and negative values denoting convex (ridges) and concave shapes (valleys).

We computed the slope and aspect using the *r.slope.aspect* module based on the standard polynomial approximation using 3×3 window, see Eq. (4.9) to (4.14). When computing curvature, we use a more flexible method with a larger window (21×21 grid cells) to capture the larger scale morphology associated with the main valleys and ridges at our study site. This method is implemented in the module *r.param.scale* and we used this module to compute profile and tangential curvature. To visualize the variability in the curvature values we used a dedicated color ramp with divergent scheme:

```
r.slope.aspect elevation=scan aspect=scan_aspect \
    slope=scan_slope
r.param.scale input=scan output=scan_pcurv size=21 method=longc
r.param.scale input=scan output=scan_tcurv size=21 method=crosc
r.colors map=scan_pcurv color=curvature
r.colors map=scan_tcurv color=curvature
```

To build a new road with gentle slopes we constructed a series of switchbacks with low areas raised on embankments. As a result there are steep slopes beside the road on the faces of the embankments (Fig. 4.8). We also flattened parts of the hillside as building sites. When we analyzed the curvature of the modified landscape the flat roads were mapped as concave areas with high curvature due to the larger neighborhood size used in the analysis (Fig. 4.9).

4.2.4 Changing landforms

The new development would change the landforms creating new ridges and valleys. We explored these changes by running the *r.geomorphon* add-on module which can be installed using *g.extension*. We can tune the results with parameters such as *search* and *skip* that determine the appropriate scale of analysis. By increasing these parameters we can skip small terrain variations in order to capture larger landforms:

```
g.extension r.geomorphon
r.geomorphon dem=elevation forms=landforms search=16 skip=6
```

In the resulting landform classification the new road has formed a major ridge (Fig. 4.10).

Fig. 4.8: Slope (a, b) and aspect (c, d) in degrees computed from the scanned model before (a, c) and after (b, d) modification.

Fig. 4.9: Profile (a, b) and tangential (c, d) curvature computed from the scanned model before (a, c) and after (b, d) modification.

Fig. 4.10: Landform classification: (a) before development, (b) after development with the new elevated road classified as a ridge, (c) legend and (d) a detailed view of the modification done in Fig. 4.7.

References

Abramowitz, M. and Stegun, I. (1965). *Handbook of mathematical functions: with formulas, graphs, and mathematical tables*, volume 55. Dover publications.

Horn, B. K. P. (1981). Hillshading and the reflectance map. *Proceedings of the IEEE*, 69(1):41–47.

Jasiewicz, J. and Stepinski, T. F. (2013). Geomorphons – a pattern recognition approach to classification and mapping of landforms. *Geomorphology*, 182:147–156.

Mitasova, H. and Mitas, L. (1993). Interpolation by regularized spline with tension: I. Theory and implementation. *Mathematical Geology*, 25(6):641–655.

Mitasova, H., Mitas, L., and Harmon, R. (2005). Simultaneous spline approximation and topographic analysis for lidar elevation data in open-source GIS. *IEEE Geoscience and Remote Sensing Letters*, 2:375–379.

Olaya, V. (2009). Basic land-surface parameters. In *Geomorphometry Concepts, Software, Applications, Developments in Soil Science*, volume 33, pages 141–169. Elsevier.

Chapter 5
Surface water flow and soil erosion modeling

The topography of the Earth's surface controls the flow of water and mass over the landscape. Modifications to the surface geometry of the land redirect water and mass flows influencing ecosystems, crop growth, the built environment, and many other phenomena dependent on water and soil. We used Tangible Landscape to explore the relationship between overland flow patterns and landscape topography by manually changing the landscape model while getting near real-time feedback about changing flow patterns. We coupled Tangible Landscape with a sophisticated dam break model to investigate flood scenarios after a dam breach.

5.1 Foundations in flow modeling

Water flow over complex terrain can be described by a bivariate form of the shallow water continuity equation. The continuity equation can be coupled with momentum (Navier-Stokes) equations to simulate flooding due to dam failure.

5.1.1 Overland flow

For shallow water flow the spatial variation in velocity with respect to depth can be neglected and the overland water flow during a rainfall event can be approximated by the following bivariate continuity equation (Julien et al., 1995):

$$\frac{\partial h}{\partial t} + \nabla \cdot (h\,\mathbf{v}) = i_e \tag{5.1}$$

where:

h is the depth of overland flow in m

t is the time in s

$\mathbf{v}$ is the flow velocity vector $\mathbf{v} = (v_x, v_y)$ in m/s

i_e is the rainfall excess (rainfall − infiltration − vegetation intercept) in m/s.

If we assume that the dynamic friction slope of the water surface can be approximated by the static bare ground slope the flow velocity can be estimated by the Manning's equation:

$$\mathbf{v} = \frac{1}{n} h^{2/3} |\mathbf{s}|^{1/2} \mathbf{s_0} \tag{5.2}$$

where:

n is the Manning's coefficient expressed here with units $\mathrm{m}^{-\frac{1}{3}}\mathrm{s}$

$\mathbf{s}$ is the negative elevation gradient $-\nabla z = -(f_x, f_y)$ of the bare ground surface

$|\mathbf{s}|$ is the magnitude of elevation gradient (slope, see subsection 4.1.3)

$\mathbf{s_0}$ is the unit vector in the flow direction (the negative elevation gradient direction).

For steady state water flow and a steady rainfall excess rate the continuity equation has the following form:

$$\partial h / \partial t = 0 \quad \longrightarrow \quad \nabla \cdot (h\ \mathbf{v}) = i_e \tag{5.3}$$

We introduce a diffusion-like term proportional to $\nabla^2[h^{5/3}]$ to incorporate the diffusive wave effects in at least an approximate way. The spatially distributed, steady overland water flow is then expressed as follows:

$$-\frac{\varepsilon}{2} \nabla^2 [h^{5/3})] + \nabla \cdot (h\ \mathbf{v}) = i_e \tag{5.4}$$

where ε is a spatially variable diffusion coefficient. The diffusion term, which depends on $h^{5/3}$ instead of h, makes the equation (Eq. 5.4) linear in the function $h^{5/3}$ which enables us to solve it by the Green's function method using a stochastic technique referred to as path sampling method (Mitasova et al., 2004).

The path sampling method is based on the duality between the particle and field representation of spatially distributed phenomena. In this concept the density of particles in a space defines a field and vice versa, i.e. a field is represented by particles with a corresponding spatial distribution. Using this duality, processes can be modeled as the evolution of fields or the evolution of spatially distributed particles (Mitasova et al., 2004).

The flow evolution (accumulation process) can be also interpreted as an approximation of a dynamical solution for shallow water flow, in which velocity is mostly controlled by terrain slope and surface roughness rather than water

depth and friction slope. Therefore the change of velocity over time at a given location is negligible. The robustness of the path sampling method enables us to simulate complex, spatially variable conditions and efficiently explore flow patterns for a wide range of landscape configurations. This method is implemented in GRASS GIS as the module *r.sim.water*.

5.1.2 Dam break modeling

Surface water flow and flooding due to a dam breaking are also represented by the shallow water flow equation, but in this application we assume that there is no rainfall and the only source is water from the reservoir. In order to ensure a realistic representation given a large mass of moving water the shallow water continuity equation:

$$\frac{\partial h}{\partial t} + \nabla \cdot (h\,\mathbf{v}) = 0 \tag{5.5}$$

is coupled with the momentum (Navier-Stokes) equation of fluid motion (Cannata and Marzocchi, 2012):

$$\frac{\partial h v_x}{\partial t} + \frac{\partial h v_x^2}{\partial x} + \frac{\partial h v_x v_y}{\partial y} = S_x \tag{5.6}$$

$$\frac{\partial h v_y}{\partial t} + \frac{\partial h v_y v_x}{\partial x} + \frac{\partial h v_y^2}{\partial y} = S_y \tag{5.7}$$

where $\mathbf{S} = (S_x, S_y)$ is the source vector (water flowing from the lake):

$$S_x = -gh\left(\frac{\partial z_w}{\partial x} + \frac{n^2 v_x |\mathbf{v}|}{h^{4/3}}\right) \qquad S_y = -gh\left(\frac{\partial z_w}{\partial y} + \frac{n^2 v_y |\mathbf{v}|}{h^{4/3}}\right) \tag{5.8}$$

and

- g is the gravitational acceleration in $\mathrm{m\,s^{-2}}$
- z_w is the water level expressed as elevation above sea level in m
- n is the Manning's roughness coefficient expressed here with units $\mathrm{m^{-\frac{1}{3}}s}$
- $|\mathbf{v}|$ is the flow velocity magnitude $|\mathbf{v}| = (v_x^2 + v_y^2)^{1/2}$ in m/s.

The resulting partial differential equations are hyperbolic and non-linear and therefore must be solved numerically. The finite volume method with an upwind conservative scheme proposed by Ying et al. (2004) is used to solve the equations on a regular grid which makes the model suitable to GIS-based implementation. Green's theorem is used to obtain the discrete equations. The numerical method explicitly solves the governing equations in two separate steps. First, the continuity equation 5.5 is evaluated deriving the water depth at time $t + \Delta t$ and later on these values are used in the source term (Eq. 5.8) to solve the momentum equations 5.6, 5.7 and evaluate the flow velocities

at time $t + \Delta t$. More details about the numerical solution can be found in Cannata and Marzocchi (2012). The model is implemented in GRASS GIS as an add-on module *r.damflood*.

5.1.3 Sediment transport, erosion, and deposition

The *Simplified Erosion/Deposition model* (*SED*) model estimates sediment transport across a complex landscape and the resulting pattern of erosion and deposition using the idea originally proposed by Moore and Burch (1986). The model assumes that the sediment flow can be approximated by sediment transport capacity and the net erosion and deposition is transport capacity limited. It combines the Universal Soil Loss Equation parameters and upslope contributing area per unit width A to estimate the sediment flow T:

$$T \approx RKCPA^m(\sin\beta)^n \tag{5.9}$$

where:

- T is the sediment flow rate in $\mathrm{kg\,m^{-1}\,s^{-1}}$
- R is the rainfall factor in $\mathrm{MJ\,mm\,(ha\,hr\,yr)^{-1}}$
- K is the soil erodibility factor in $\mathrm{ton\,ha\,hr\,(ha\,MJ\,mm)^{-1}}$
- A is the upslope contributing area per unit width in m
- β is the steepest slope angle
- C is the dimensionless land-cover factor
- P is the dimensionless prevention-measures factor.

The net erosion/deposition D in $\mathrm{kg\,m^{-2}s^{-1}}$ is then computed as a divergence of sediment flow:

$$D = \nabla \cdot (T\mathbf{s_0}) = \frac{\partial(T\cos\alpha)}{\partial x} + \frac{\partial(T\sin\alpha)}{\partial y} \tag{5.10}$$

where $\mathbf{s_0} = (\cos\alpha, \sin\alpha)$ is the unit vector of the steepest slope (flow) direction (direction of the negative gradient $-\nabla z$) given by the aspect angle α (see Section 4.1.3). The exponents m, n control the relative influence of the water and slope terms and reflect the impact of different types of flow. The typical range of values is $m = 1.0-1.6$, $n = 1.0-1.3$ with the higher values reflecting the pattern for prevailing rill erosion with more turbulent flow when erosion sharply increases with the amount of water. Lower exponent values close to $m = n = 1$ better reflect the pattern of the compounded, long term impact of both rill and sheet erosion and averaging over a long term sequence of large and small events.

5.2 Case study: the hydrological impact of development

The development on North Carolina State University's Centennial Campus raises concerns about the impact of stormwater runoff and erosion during construction. We used Tangible Landscape to explore changes in overland flow and erosion patterns caused by potential construction site grading and to provide feedback for the design of stormwater and erosion control measures.

In this case study we worked with the same area and physical model used in Section 4.2.1. To familiarize ourselves with the basic hydrologic conditions of this site we computed flow accumulation and delineated watershed boundaries. First, we set the computational region to match our study area and the resolution to 1 m and then we ran the least cost path flow tracing implemented in the module *r.watershed* to derive the flow accumulation raster and watersheds. We set the threshold for the approximate size of the watershed areas to 1000 grid cells and we converted the areas to a vector representation so that we could display the watershed boundaries:

```
g.region n=223765 s=223015 e=638450 w=637700 res=1 -p
r.watershed elevation=dem accumulation=flow_accum \
    basin=watersheds threshold=1000
r.to.vect input=watersheds output=watersheds type=area
```

The resulting map that combines the flow accumulation raster with the vector representation of watershed boundaries and contours was then projected over the model in Fig. 5.1a.

5.2.1 The impact of development on surface water flow

To explore the potential impact of new development we graded sites for new apartment buildings with flat slopes and constructed an access road (see Fig. 4.7d). We carved a narrow culvert to allow water to flow under the road. By calculating flow accumulation and watersheds for the modified topography (Fig. 5.1b) we could immediately see that the watershed boundaries in the developed area changed significantly due primarily to the new road acting as an artificial watershed boundary.

Flow accumulation derived by the module *r.watershed* is based on the least cost path algorithm and is not designed to represent water depth and pooling of water in depressions. Therefore, we used the shallow water flow model implemented in the module *r.sim.water* to simulate the overland flow depth during a storm event for the current conditions and after the grading. We first derived the components of elevation surface gradient $\nabla z = (f_x, f_x)$ (parameters dx, dy in the module *r.slope.aspect*) and then ran the water flow simulation with the rainfall excess value of 150 mm/hr:

```
r.slope.aspect elevation=scan dx=scan_dx dy=scan_dy
r.sim.water elevation=scan dx=scan_dx dy=scan_dy \
    rain_value=150 depth=depth
```

(a)

(b)

Fig. 5.1: Flow accumulation and watershed boundaries computed with *r.watershed* before and after the terrain modifications shown in Fig. 4.7.

The resulting water depth maps were projected over the model with the initial terrain conditions and then in near real time as the terrain was modified with the final water depth pattern shown in Fig. 5.2b. Note that we have used an extremely high value of rainfall excess to provide rapid maps of water depth as the model was modified. Once we have the final design we can run the simulation with a selected storm and obtain realistic water depth estimates.

We can also run the module *r.sim.water* in a dynamic mode (-*t* flag) and project the evolution of water depth during the storm event as an animation using the GRASS GIS animation tool.

(a) (b)

(c)

Fig. 5.2: Water depth computed with *r.sim.water* (a) before and (b) after the terrain modifications shown in Fig. 4.7. A detailed view of the modified water flow is shown in (c).

5.2.2 The impact of grading on erosion and deposition pattern

We also explored the impact of grading on sediment transport and erosion/deposition patterns using the SED model. We assumed a uniform rainfall factor $R = 270$ in (hundreds of ft-tonf.in)/(acre.hr.year) as well as uniform soils and land cover with the soil erodibility factor $K = 0.2$ and the land cover factor $C = 0.01$ (grass, no trees). The topographic factor exponents were set to $m = n = 1$ to represent prevailing sheet flow. Since we used uniform values for the R-, K- and C-factors, the pattern of erosion and deposition depends

solely upon the slope of the terrain and the upslope contributing area. In the following workflow we first computed the slope and aspect maps of the scanned model. Then we computed flow accumulation and combined it with the slope using map algebra to estimate the topographic factor and sediment transport capacity (see Eq. 5.9).[1] Finally, we computed the divergence of the sediment flow vector field with the add-on module *r.divergence* to generate the erosion-deposition map:

```
r.slope.aspect elevation=scan aspect=scan_aspect \
    slope=scan_slope
r.flow elevation=scan flowaccumulation=scan_flowacc
r.mapcalc "scan_sedflow = 270. * 0.2 * 0.01 * scan_flowacc * \
    sin(scan_slope)"
r.divergence magnitude=scan_sedflow direction=scan_aspect \
    output=scan_usped
r.colors map=scan_usped rules=color_erdep.txt
```

The resulting erosion-deposition map is projected over the model in Fig. 5.3.

5.3 Case study: dam breach

The Lake Raleigh dam located within our Centennial Campus study area broke in September of 1996 after Hurricane Fran. In this case study we simulated how flood water would spread across the current landscape if the dam was breached again. We also modified the terrain to explore how different morphologies would influence the spread of flooding. We used the add-on *r.damflood*, which was designed specifically to simulate dam breaks and analyze the subsequent flooding.

5.3.1 Site description and input data processing

Fig. 5.4 shows the terrain and the most recent orthophotograph of our Lake Raleigh study area including the dam. The stream flowing out from the lake merges with the stream coming from the south in the area just below the dam, which is actively used for recreation providing space for disc golf, running and cycling (Fig. 5.5a).

The dam break simulation requires several input layers including the elevation raster that describes the lake bathymetry and the surrounding topography, the depth of the lake water, the Manning's roughness coefficient, and the geometry of the dam break. The elevation raster was created by fusing the latest DEM based on the 2014 lidar survey with a DEM from the 2001 survey which we used as a proxy for the lake bathymetry. The 2001 survey captured

[1] For a general case when m, n and resolution are not set to 1 see the erosion modeling tutorial: `http://ncsu-osgeorel.github.io/erosion-modeling-tutorial/grassgis.html`

(a) (b)

(c)

Fig. 5.3: Soil erosion and deposition maps draped over the model (a) before and (b) after grading shown in Fig. 4.7. Detailed view of the erosion-deposition pattern is in (c).

Fig. 5.4: An overview of Lake Raleigh's surroundings with the dam highlighted in red. The digital elevation model on which the orthophoto is draped was created by fusing the latest 2014 data with the lake bathymetry data from 2001.

the topography while the lake was drained after the dam failure in 1996 and we considered the 2001 DEM in the lake area a suitable approximation of the current lake bathymetry.

The lake depth raster was computed with the module *r.lake* using the fused DEM as an input. This module fills an area with water for a designated elevation level starting from a given seed point and outputs a map of the water depth for the flooded area. We determined that the current water level elevation for the Lake Raleigh was 85 m by querying the DEM along the outline of the lake. We selected an arbitrary point inside the lake as a seed point. While *r.lake* assigns null values to the cells outside of the filled area, *r.damflood* requires zero values outside of the lake, so we converted null values to zeros with the *r.null* module and limited the flooded area to the lake mask:

```
g.region n=224026 s=223007 e=639480 w=637798 res=3 -p
r.lake elevation=lake_bottom_dem water_level=85 \
    lake=lake_depth_tmp coordinates=638792,223659
r.null map=lake_depth_tmp null=0
r.mapcalc "lake_depth = if(lake_mask, lake_depth_tmp, 0)"
```

The Manning's roughness coefficient influences the speed of water. It depends on landcover and varies between 0 and 1 with higher values resulting in slower flows. To derive Manning's coefficients we could for example reclassify a landcover map. In our example for simplicity's sake we created a raster map with a uniform Manning's value:

```
r.mapcalc "manning = 0.01"
```

Finally, the module *r.damflood* requires a dam breach raster which represents the height of the breach from the top of the dam. To obtain this raster we scanned the model with the dam. Then we carved a breach into the dam and rescanned the model. We computed the difference using the function defined in code snippet 4.2.2. We used a vertical limit of 1 m to avoid detecting differences due to noise from scanning:

```
adjust_scan('scan_before', 'scan_after', 'scan_adjusted')
gscript.mapcalc("breach = if(scan_before - scan_adjusted > 1,
    scan_before - scan_adjusted, 0)")
```

5.3.2 The impact of the road on flooding

For our first flood simulation we used the current conditions. A major road with a culvert crosses the stream below the dam. We carved a culvert into the sand model by removing a narrow channel of sand from the road and scanned this model to provide input for our simulation.

Since our sand model does not incorporate the lake bathymetry we had to combine the fused DEM that includes the bathymetry with the scan. We then ran the *r.damflood* module with the input map layers and set the simulation parameters such as the length of the simulation, the time step for creating

(a) (b)

Fig. 5.5: (a) The current conditions with a stream and a greenway below the dam; (b) a scenario in which the road has been removed.

output maps, and the computational time step. Since the simulation is computationally intensive it may take several minutes to compute depending on the settings and resolution.

```
r.mapcalc "scan_combined = if(lake_mask > 0, lake_bottom_dem, \
    scan_adjusted)"
r.damflood elev=scan_combined lake=lake_depth dambreak=breach \
    manning=manning h=flood timestep=0.1 tstop=1000 deltat=10
```

The series of output raster maps generated by the simulation can be registered in the GRASS GIS Temporal Framework and then visualized using module *g.gui.animation*. The results for the current conditions are shown in Fig. 5.6. Clearly, the road creates a significant obstacle for the spread of water and causes pooling below the dam. Contained by the road the water floods upstream areas to the west.

Fig. 5.6: The flood simulation with current conditions

In the next simulation we were interested in the conditions before the road was built in 2004. We simply removed the road (Fig. 5.5b) and then rescanned and recomputed the simulation (Fig. 5.7). Removing the road changes the flow pattern and as a result water spreads freely at lower depth thus impacting a larger area, but potentially reducing risk.

Fig. 5.7: The flood simulation after the road has been removed

References

Cannata, M. and Marzocchi, R. (2012). Two-dimensional dam break flooding simulation: A GIS-embedded approach. *Natural Hazards*, 61(3):1143–1159.

Julien, P. Y., Saghafian, B., and Ogden, F. L. (1995). Raster-based hydrologic modelling of spatially-varied surface runoff. *Water Resources Bulletin*, 31(3):523–536.

Mitasova, H., Thaxton, C., Hofierka, J., McLaughlin, R., Moore, A., and Mitas, L. (2004). Path sampling method for modeling overland water flow, sediment transport, and short term terrain evolution in Open Source GIS. *Developments in Water Science*, 55:1479–1490.

Moore, I. D. and Burch, G. J. (1986). Physical basis of the length-slope factor in the Universal Soil Loss Equation. *Soil Science Society of America journal*, 50:1294–1298.

Ying, X., Khan, A. A., and Wang, S. S. (2004). Upwind conservative scheme for the Saint Venant equations. *Journal of hydraulic engineering*, 130(10):977–987.

Chapter 6
Viewshed analysis

Viewshed (visibility) analysis is used in many different fields for both practical and aesthetic applications. It can play an important role when planning new buildings or roads especially in urban settings where obstructed views may raise safety concerns. In recreation areas views of beautiful landscapes are highly valued and protected with great passion. Visibility analysis is also crucial when planning location of monitoring cameras or communication towers in order to maximize coverage. With the increasing availability of high-resolution DEMs and DSMs derived from lidar visibility analysis is becoming more accurate, broading the range of its applications. We used Tangible Landscape to analyze viewsheds on North Carolina State University's (NCSU) Centennial Campus from different observer positions and explored how future development would affect the viewsheds. We introduced object recognition to collaboratively designate observer positions.

6.1 Line of sight analysis

Line-of-sight analysis is used to map visible areas. If the line of sight between point A and point B does not intersect the terrain then these points are mutually visible. More precisely, with the slope of the line between A and B defined as $(z_B - z_A)/d_{AB}$ where z is the height and d is the horizontal distance, points A and B are considered visible to each other if the line does not cross any location C such that the slope of AC is larger than the slope of AB. The viewshed of A on a raster DEM is the set of all cells of the DEM that are visible from A. The height of the observer A above terrain has significant influence on the viewshed extent. The observer height can be changed to model the view of a standing person, the view from a multi-story building, or the view from the top of a cell tower. The resulting viewshed can be represented for example as a binary raster (containing only 0 and 1s) or a raster with the values of vertical angles with regard to the viewpoint.

Viewsheds from multiple locations can be computed and combined as the sum of binary viewsheds into a cumulative viewshed (Wheatley, 1995) that allows us to assess the overall visibility of a place from selected locations. The concepts of visibility analysis have been applied to diverse geospatial applications. For example landform identification using geomorphons combines visibility analysis with computer vision (Jasiewicz and Stepinski, 2013) (see Section 4.1.3). Several terrain relief visualization techniques take advantage of visibility analysis to highlight subtle terrain features. For example sky-view factor – the visible part of the sky unobscured by relief – can be used to render topographic relief as if it was diffusely illuminated in order to clearly visualize the relative height of features. Sky-view factor uses openness as a proxy for uniform diffuse illumination (Zakšek et al., 2011). It is implemented in GRASS GIS as the add-on module *r.skyview*. Haverkort et al. (2009) describe an efficient method implemented in the module *r.viewshed* for computing viewsheds on a raster DEM using a technique called line sweeping.

Fig. 6.1: Line of sight analysis: point B is not visible from point A because the line of sight between A and B has a smaller slope $\tan \alpha_{AB}$ than the line of sight between A and C.

6.2 Case study: viewsheds around Lake Raleigh

We used object recognition with Tangible Landscape to explore different viewsheds on NCSU's Centennial Campus by placing markers representing observer positions on a physical model of the landscape. We also evaluated how changes to the topography, the forest canopy, and the buildings would impact these viewsheds.

6.2.1 Site description and model

Our study area shown in Fig. 6.2 is the Lake Raleigh landscape on NCSU's Centennial Campus. Key features of the landscape include Lake Raleigh

Woods in the southwest, the Chancellor's house and State Club in the south, and the Hunt Library in the northwest.

We CNC routed a base model and a mold of the landscape surface from MDF based on a lidar-derived DEM and DSM. Both models are approximately 1 : 2400 scale without vertical exaggeration. We used the base and the mold to cast a malleable layer of polymer enriched sand representing the forest canopy and buildings on top of the terrain model.

Fig. 6.2: Lake Raleigh study site: a lidar-based DSM with an orthophoto draped over the surface

6.2.2 Identifying view points

We modeled the views from different locations by placing markers on the model for Tangible Landscape to detect and recognize. The position of each marker was registered as a vector point and its coordinates were then used as an input for automated computation of viewshed from this location.

Given the complex 3D form and steep slopes of some terrains it may be hard to place markers on a rigid model such as a 3D printed or CNC routed terrain. If the model is coated with magnetic paint then a magnetized marker will hold to the slopes, normal to the surface (see Fig. 3.7). If the model has been sculpted or cast in polymeric sand then markers can simply be stuck vertically into the sand.

There are multiple ways to detect markers. The most reliable approach is to detect the markers based on the difference between a base scan of the terrain model without the markers and the scan with the markers placed on the model. We computed the difference using the function defined in the code snippet in Section 4.2.2. A threshold value for the difference in elevation must be set in order to distinguish markers from noise. The threshold can be

specified in millimeters and since we know the horizontal scale and vertical exaggeration during the processing it can be automatically converted into the real world elevation difference required in the following code snippet. Areas with difference values above the threshold are vectorized as points for viewshed analysis as follows. We first convert the areas from raster to vector areas (polygons) using the module *r.to.vect* and then extract the area centroids as vector points using the module *v.to.points*.

```
adjust_scan('scan_before', 'scan_after', 'scan_adjusted')
gscript.mapcalc("change = if(scan_adjusted - scan_before >
    threshold, 1, null())")
gscript.run_command('r.to.vect', flags='st', input='change',
    output='change_vector', type='area')
gscript.run_command('v.to.points', flags='t',
    input='change_vector', type='centroid', output='markers')
```

6.2.3 Visibility analysis on DSM using markers

We explored viewsheds from different points of interests such as the Chancellor's house, the North Shore neighborhood, the Hunt Library's observation deck, the lakeside pier, and the middle of the lake. The viewsheds were computed on the lidar-derived DSM. To identify the markers we first scanned the model and saved the scanned raster. We continued scanning while placing the markers at the locations of interest (Fig. 6.3). The markers were then identified and stored as vector points. We used the following workflow to compute the viewsheds and set the desired color scheme. In this case visible areas are highlighted in yellow.

```
coordinates = gscript.read_command('v.out.ascii',
    input='change_vector', separator=',').strip().splitlines()
for i, line in coordinates:
   output = 'viewshed_%s' % i
   coordinate = [float(c) for c in line.split(',')[0:2]]
   gscript.run_command('r.viewshed', flags='b',
       input='elevation', output=output,
       observer_elevation=1.75, coordinates=coordinate)
   gscript.run_command('r.null', map=output, setnull=0)
   gscript.write_command('r.colors', map=output, rules='-',
       stdin='1 yellow')
```

The viewsheds computed from different marker positions are shown in Fig. 6.4. The best views of the lake are from the Hunt Library's observation deck and the lakeside pier.

6.2.4 Modeling viewsheds from a new building

In the previous section we computed viewsheds on the lidar-derived DSM. In this section we used the scanned surface to model viewsheds on a modified landscape. We explored a scenario in which we built a hotel on the

(a) (b)

Fig. 6.3: Placing a marker to identify a viewpoint.

(a) (b)

(c) (d)

Fig. 6.4: Viewsheds computed on lidar-based DSM from viewpoints at (a) the North Shore neighborhood, (b) Hunt Library's observation deck, (c) a lakeside pier, (d) and a canoe on the lake.

southeastern side of the lake and assessed the viewshed from the new hotel (Fig. 6.5a).

We began by sculpting the new building in sand. Then we scanned the modified sand surface and saved the scan. We placed a marker that was automatically detected and digitized as a viewpoint. We computed the viewshed using the base scan rather than the newer scan with the marker because the marker would act as an obstacle blocking the observer's view. Fig. 6.5c shows the viewshed from the southwest side of the new hotel. The hotel is oriented so that it has nice views of the lake. Since the hotel is a tall building it is visible from many places around the lake such as Hunt Library (Fig. 6.5d).

Fig. 6.5: Modeling viewsheds from a new building: (a) building a new hotel, (b) digitizing a viewpoint, and evaluating viewsheds from (c) the hotel and (d) the library.

References

Haverkort, H., Toma, L., and Zhuang, Y. (2009). Computing visibility on terrains in external memory. *J. Exp. Algorithmics*, 13:5:1.5–5:1.23.

Jasiewicz, J. and Stepinski, T. F. (2013). Geomorphons – a pattern recognition approach to classification and mapping of landforms. *Geomorphology*, 182:147–156.

Wheatley, D. (1995). Cumulative viewshed analysis: a GIS-based method for investigating intervisibility, and its archaeological application. *Archaeology and geographical information systems: a European perspective*, pages 171–185.

Zakšek, K., Oštir, K., and Kokalj, v. (2011). Sky-View Factor as a Relief Visualization Technique. *Remote Sensing*, 3(2):398–415.

Chapter 7
Trail planning

The design of a walking or hiking trail is based on fine scale topographic conditions and varied criteria specific to the particular context such as aesthetics, views, construction cost, and environmental sensitivity. As a result trail planning is typically a product of expert knowledge, field surveys, and creative design decisions – often made on site. However, when high resolution data is available geospatial modeling can be used to identify routes optimized for travel time and suitability. To design trails with Tangible Landscape we can hand place waypoints on a physical model and then the optimal network connecting the waypoints is computed in near real-time. This approach – hand placing tangible waypoints and computationally networking the waypoints – combines creative, collaborative decision making with mathematical optimization. In this chapter we explain the theory and methodology for designing trails with Tangible Landscape and then discuss a case study, the design of hiking trail scenarios for Lake Raleigh Woods, North Carolina.

7.1 Trail design methodology

Our approach for designing trail networks with Tangible Landscape combines the creative identification and siting of key waypoints like trailheads and scenic spots with computationally optimized routing between these points. With Tangible Landscape we can place tangible markers by hand to automatically digitize waypoints. This allows us to work intuitively, feel the slopes and curvature with our hands, and easily collaborate.

To find the optimal routes we compute the least cost path between pairs of waypoints over the terrain and a cost surface as a function of walking energetics. First we create a cost surface that represents friction – the additional time required to cross a cell. Next we designate waypoints, points through which we want the trail to pass. For each unique pair of waypoints we use the *r.walk* module to compute the anisotropic cumulative cost of walking

between these points across the terrain and the friction surface based on Naismith's rule for hiking times. Then we compute the least cost path between those points across this cumulative cost surface. All of the least cost paths are combined into a network of potential routes. Finally we solve the traveling salesman problem to identify the best route across this network.

7.1.1 Least cost path analysis

With least cost path analysis we can determine the most cost-effective route between a source and destinations on a cost surface. The cost surface is a raster representing cost for traversing a cell and can be derived as a function of distance, slope, land cover or other relevant criteria. To find the least cost path between locations we need to compute a cumulative cost surface where each cell contains the lowest cumulative cost of traversing the space between each cell and the specified location. We also need to generate a movement direction raster tracing the movements that created the cumulative cost surface.

First we compute the walking cost raster using Naismith's rule for walking times (Naismith, 1892) with further adjustments (Aitken, 1977; Langmuir, 1984) of the cost for specific slope intervals. Considering only the elevation surface the cost to walk across a grid cell, expressed as time T in seconds is computed as follows:

$$T = a \cdot \Delta S + b \cdot \Delta H_u + c \cdot \Delta H_{md} + d \cdot \Delta H_{sd} \tag{7.1}$$

where:

- ΔS is the horizontal distance in m
- ΔH is the height difference in m
- a is the time in seconds it takes to walk for 1 meter on flat surface
- b is the additional walking time in seconds, per meter of elevation gain ΔH_u on uphill slopes
- c is the additional walking time in seconds, per meter of elevation loss ΔH_{md} on moderate downhill slopes
- d is the additional walking time in seconds, per meter of elevation loss ΔH_{sd} on steep downhill slopes

Up to a specific slope value threshold, walking downhill is faster; after that it becomes more difficult and adds to the time needed to cross the cell. The slope value threshold (slope factor) derived from experiments is -0.2125 corresponding to 12° downslope ($\tan 12° = 0.2125$).

Taking into account land cover conditions the total cost T_{total} in seconds is estimated as a linear combination of movement and friction costs using the dimensionless weight λ:

$$T_{\text{total}} = T + \lambda \cdot F \cdot \Delta S \tag{7.2}$$

where F is the friction in s/m. It represents the additional time in seconds that it takes to walk one meter within a given cell due to its land cover conditions. The friction map may simply be a landcover map that has been recoded as time costs. If landcover data is not available at an appropriate resolution or if there are other important factors that need to be considered, the friction map can be created through map overlay analysis. Parameters such as soils, hydrology, roads, and existing trails can then be incorporated into the cost surface using the friction map.

We used the total cost surface given by Eq. 7.2 to compute a cumulative cost surface where each cell contains the lowest cumulative cost of traversing the space between each cell in the region and the target location. The movement direction raster is also generated tracing the movements that created the cumulative cost surface. We can then compute least cost path between any point in the study area and the target location based on the cumulative cost raster and movement directions raster.

7.1.2 Network analysis

Using least cost path analysis it is possible to create many combinations of routes between the given waypoints. This can result in a large number of connections and potential trails. We can use network analysis to find a trail loop that goes through all given waypoints in an optimal order, avoiding connections with high costs (Fig. 7.1). The solution for this well-known optimization problem – the traveling salesman problem (TSP)[1] – is implemented in the module *v.net.salesman*, which uses a heuristic algorithm yielding a good solution in a reasonable amount of time.

7.1.3 Trail slope extraction

When planning trails we are typically interested in the average and maximum slope of a trail. However, the extracted values are the slope values in the direction of steepest slope rather than in the direction of the trail. To compute that, we need to know the direction of the trail, which can be derived from its vector representation. The slope along the trail β_t can then be computed as the steepest slope β (see Eq. 4.5) multiplied by the cosine of the angle between the direction of the trail α_t and the steepest slope direction α (i.e. aspect, see Eq. 4.6):

$$\tan \beta_t = \tan \beta \cos(\alpha - \alpha_t) \tag{7.3}$$

The difference is visible in Fig. 7.2 where part of a trail along a contour line has very low slope values along the trail, but has higher slope values in the direction of the steepest slope. Both types of slope computation can

[1] See: `https://en.wikipedia.org/wiki/Travelling_salesman_problem`

Fig. 7.1: The TSP applied to a network of potential trails. Black lines represent potential routes. The red line is the most cost-effective combination of routes that connects all of the waypoints.

be useful depending upon our objectives. When building and maintaining a trail we may be interested in the steepness of the terrain in that area, while a hiker may be more interested in the directional slope of a trail.

7.2 Case study: designing a recreational trail

We used Tangible Landscape to design alternative trail scenarios for Lake Raleigh Woods on North Carolina State University's Centennial Campus. This old growth woodland on the south shore of Lake Raleigh is rich in biodiversity with approximately 200 species of vascular plants and regionally important stands of mesic mixed hardwood and dry-mesic oak-hickory forest (Blank et al., 2010). Informal hiking and mountain biking trails have caused significant disturbance in the woods and erosion along the lakefront. A formal trail system could concentrate people on low-impact routes and thus conserve the landscape while enhancing opportunities for recreation.

7.2.1 Input data processing

We derived a bare earth DEM and DSM with buildings and vegetation from a lidar survey acquired in 2013. Then we CNC routed 1 : 1500 scale models

(a) (b)

Fig. 7.2: The difference between the trail slope extracted (a) in the direction of the steepest slope and (b) in the direction of the trail (in degrees).

of the DEM, the DSM, and their inverses (see 3.4.4). We used the inverse models as molds for casting polymeric sand models.

We created the friction map – a required input for computing the cost surface – by computing the average friction cost for each cell from a set of friction maps based on the recoded rasterized maps of buildings, hydrology, floodplains, roads, soils, and trails with *r.series* using the average function.

First we converted all of the necessary vector maps to raster. Before converting linear features we set the resolution to 3 meters in order to give these features reasonable widths.

```
g.region n=224134 s=223005 w=637732 e=639085 res=1 -p

v.to.rast input=lake_raleigh output=lake_raleigh use=val
v.to.rast input=chancellors output=chancellors use=val
v.to.rast input=buildings output=buildings use=val
v.to.rast input=floodplain output=floodplain use=val
v.to.rast input=hydrology_areas output=hydrology_areas use=val
v.to.rast input=trail_areas output=trails use=val
v.to.rast input=roads output=roads use=val

g.region res=3
v.to.rast input=greenways output=greenways use=val
v.to.rast input=streets output=streets use=val
v.to.rast input=hydrology output=hydrology use=val
g.region res=1
```

In order to limit processing to publicly accessible areas we made a composite map of inaccessible areas by combining maps of private land, buildings,

and the lake using *r.mapcalc*[2] and then created an inverted mask from this map.

```
r.mapcalc "masking = if((buildings ||| chancellors ||| \
    lake_raleigh), 1, null())"
r.mask -i raster=masking
```

Next, we recoded all maps as friction (time penalties – additional time it takes to walk one meter), and then overlaid the maps using *r.mapcalc* and *r.series*.

```
r.mapcalc "hydro = if(isnull(hydrology_areas ||| hydrology), \
    1, 10)"
r.mapcalc "flood = if(isnull(floodplain), 1, 10)"
r.mapcalc "paths = if(isnull(trails ||| greenways), 10, 1)"
r.mapcalc "transit = if(isnull(streets ||| roads), 1, 10)"

r.series input=flood,hydro,paths,transit,soils output=friction \
    method=average
r.colors -n map=friction color=ryg
r.mask -r
```

The resulting friction map is shown in Fig. 7.3.

7.2.2 Computing the trail using the least cost path

In our case study we computed least cost paths between waypoints to create a trail network. Least cost paths were computed for each combination of two points. The following code snippet shows how such a network can be generated. For each waypoint we called the function `trail` defined below, which computes a raster representing the cumulative cost (time) from this waypoint and then multiple least cost paths from the remaining waypoints.

```
def trail(elevation, friction, walk_coeff, lambda_,
    slope_factor, point_from, points_to, vect_paths):
   """Computes least cost path to multiple locations based on
       walking time"""
   gscript.run_command('r.walk', flags='k',
       elevation=elevation, friction=friction, lambda_=lambda_,
       walk_coeff=walk_coeff, slope_factor=slope_factor,
       start_coordinates=point_from,
       stop_coordinates=points_to, output='tmp_walk',
       outdir='tmp_walk_dir')

   for i in range(len(points_to)):
      gscript.run_command('r.drain', flags='d',
          input='tmp_walk', direction='tmp_walk_dir',
          output='tmp_drain', drain=vect_paths[i],
          start_coordinates=points_to[i], overwrite=True)
```

[2] Note that | | | is a symbol for logical OR which ignores NULL values and treats them as logical false.

Fig. 7.3: The friction map in seconds per meter created by combining layers of buildings, hydrology, floodplains, roads, and trails.

```
  # remove temporary maps
  gscript.run_command('g.remove', type=['raster', 'vector'],
      name=['tmp_walk', 'tmp_walk_dir', 'tmp_drain'],
      flags='f')
```

This is repeated for all waypoints except the last.

```
def trails_combinations(elevation, friction, walk_coeff,
    lambda_, slope_factor, points, vector_routes):
   import itertools

   coordinates = gscript.read_command('v.out.ascii',
       input=points, format='point', separator=',').strip()
   coords_list = []
   for coords in coordinates.split():
      coords_list.append(coords.split(',')[:2])

   combinations = itertools.combinations(coords_list, 2)
   combinations = [list(group) for k, group in
       itertools.groupby(combinations, key=lambda x: x[0])]

   i = k = 0
   vector_routes_list = []
   for points in combinations:
```

```
        i += 1
        point_from = ','.join(points[0][0])
        points_to = [','.join(pair[1]) for pair in points]
        vector_routes_list_drain = []
        for each in points_to:
           vector_routes_list_drain.append('route_path_' +
               str(k))
           k += 1
        vector_routes_list.extend(vector_routes_list_drain)

        trail(elevation, friction, walk_coeff, lambda_,
            slope_factor, point_from, points_to,
            vector_routes_list_drain)

    gscript.run_command('v.patch', input=vector_routes_list,
        output=vector_routes)
```

The path, however, is only computed for combinations that were not covered in previous steps in order to reduce computation time. This means that the least cost path is only computed in one direction. The direction matters because the speed of walking uphill and downhill differs. Therefore, the walking coefficients and slope factor of *r.walk* should be adjusted to affect uphill and downhill speed equally. This can be achieved by setting the slope factor to zero and using the same absolute number for coefficients b and d.

7.2.3 Finding the optimal trail

Using least cost path analysis we created all of the combinations of routes between waypoints. This can result in a large number of connections and potential trails. Therefore we used network analysis to find a trail loop that goes through all our waypoints in an optimal order avoiding connections with high costs. The solution for this well-known optimization problem – the traveling salesman problem (TSP) – is implemented in the module *v.net.salesman*.

All GRASS GIS network analysis modules require a network vector map containing connected lines (arcs) and points (nodes). This vector map can be prepared using the module *v.net* which topologically connects the given points (our waypoints) and lines (our trail combinations). Since we wanted to route the trail through all our waypoints we found the categories of all waypoints in the network and then passed them to module *v.net.salesman*:

```
def trails_salesman(trails, points, output):
   gscript.run_command('v.net', input=trails, points=points,
       output='tmp_net', operation='connect', threshold=10)
   cats = gscript.read_command('v.category', input='tmp_net',
       layer=2, option='print').strip().split()
   gscript.run_command('v.net.salesman', input='tmp_net',
       output=output, center_cats=','.join(cats), arc_layer=1,
       node_layer=2)
   # remove temporary map
```

```
    gscript.run_command('g.remove', type='vector',
        name='tmp_net', flags='f')
```

The final computation combines above defined functions and takes the scanned DEM and the detected points as input:

```
trails_combinations('scan', friction='friction',
    walk_coeff=[0.72, 6, 0, 6], lambda_=0.5, slope_factor=0,
    points='markers', vector_routes='route_net')
trails_salesman(trails='route_net', points='markers',
    output='route_salesman')
```

Examples of final output are shown in Fig. 7.1 and 7.4.

Fig. 7.4: A trail around Lake Raleigh

7.2.4 Mapping trail slopes

The simplest way to map the slope of a trail is to rasterize the trail using *v.to.rast* and then use *r.mapcalc* to extract the slope values from a slope raster computed with the module *r.slope.aspect*:

```
r.slope.aspect elevation=dem slope=slope
v.to.rast input=trail output=raster_trail type=line use=cat
r.mapcalc "trail_slope = if(raster_trail, slope)"
```

This extracts the slope values in the direction of the steepest slope. To compute the slope in the direction of the trail we first need to rasterize its vector representation by deriving raster values from the vector direction. The slope along the trail can then be computed as the slope multiplied by the cosine of the angle between the direction of the trail and the steepest slope (i.e. aspect):

```
r.slope.aspect elevation=dem slope=slope aspect=aspect
v.to.rast input=trail output=raster_trail_dir type=line use=dir
r.mapcalc "trails_slope_dir = abs(atan(tan(slope) * cos(aspect \
    - raster_trail_dir)))"
```

(a) The viewshed from a waypoint with areas visible represented in orange

(b) The slope of the trail

Fig. 7.5: Trail analytics

7.2.5 Alternative trail scenarios

We designed the initial trail scenario for Lake Raleigh Woods by placing markers at key points of interest – potential trailheads and scenic viewpoints (for more information about marker detection see Section 6.2.2). The siting of trailheads was informed by a previous 1 : 2500 scale study of potential routes

(a) The initial route for a trail in Lake Raleigh Woods

(b) Building bridges to reroute the trail across gullies with steep slopes

(c) The updated route for the trail across the new bridges

Fig. 7.6: Iteratively designing trail scenarios for Lake Raleigh Woods

around the entire lake (Fig. 7.4). We used realtime viewshed analysis to evaluate potential viewpoints (Fig. 7.5a). To test the viewsheds we cast the DSM in polymeric sand, carved clearings in the woodland canopy for viewpoints, placed markers to designate the viewpoints, and computed the viewsheds. We also computed the slope along the trail so that we could critique the routes (Fig. 7.5b).

Our trail needed to cross three large gullies with steep slopes. In the initial trail scenario the route follows the contours and ridge lines to avoid the gullies with their challenging slopes. This route formed two loops that intersected at the mouth of the central gully near the lakefront with moderately steep slopes (Fig. 7.6a). We, however, wanted to experiment and see what a route with a single loop would look like so we built bridges across the gullies (Fig. 7.6b). The trail rerouted across the bridges to form a single, shorter loop with a lower average slope (Fig. 7.6c).

References

Aitken, R. (1977). *Wilderness Areas in Scotland*. PhD thesis, University of Aberdeen.

Blank, G., Rudder, C., Dombrowski, A., Cser, H., Lawler, M., Kollar, C., Shifflett, S. D., Fortney, J., Sullivan, R., and Canavin, J. (2010). General Management Plan for Lake Raleigh Woods. Technical report, College of Natural Resources, North Carolina State University, Raleigh.

Langmuir, E. (1984). Mountaincraft and leadership. The Scottish Sports Council, Edinburgh.

Naismith, W. W. (1892). Untitled. Scottish Mountaineering Club Journal.

Chapter 8
Solar radiation dynamics

Solar radiation (insolation) is the primary driving force for Earth's atmospheric, biophysical, and hydrologic processes. Knowing the amount of radiation at different geographic locations at different times is therefore necessary in many fields including energy production, agriculture, meteorology, ecology, and urban planning. We modeled direct solar radiation and cast shadows in an urban setting to show how different spatial configurations of buildings change the amount of sunlight available throughout the day and the year.

8.1 Solar radiation modeling

Solar radiation modeling is critical for the optimal site selection of solar power plants (Carrion et al., 2008; Janke, 2010) and for placing photovoltaic systems in often complex urban environments (Freitas et al., 2015; Hofierka and Kaňuk, 2009; Jakubiec and Reinhart, 2013). In urban design dynamic solar radiation simulations are used to study the effect of urban geometry and building configurations on solar access and shading conditions in order to get insight into the "urban canyon" phenomenon (Arnfield, 1990; Lobaccaro and Frontini, 2014).

On a global scale solar radiation depends spatially and temporally on the orientation of the Earth relative to the Sun. At a local scale solar radiation is influenced by topography (i.e. an elevation surface's inclination, orientation, and shadows), atmospheric conditions, and surface properties such as land cover. We can analyze the spatial distribution and dynamics of solar radiation and cast shadows using the equations that relate the sun position to the geometry of the terrain surface (Hofierka and Suri, 2002). The clear-sky solar radiation model applied in this chapter is based on work undertaken for the development of the European Solar Radiation Atlas (Scharmer and Greif, 2000; Page et al., 2001; Rigollier et al., 2000). We include here only the equations for direct solar radiation – beam irradiance – which is influenced by the local topography. For a complete set of equations see Hofierka and

Suri (2002). The diffuse and reflected components as well as atmospheric parameters can be also found in Rigollier et al. (2000).

First we compute position of the Sun in the sky as a function of time and location on Earth. The Sun declination angle depends on the day number (1 to 365 or 366):

$$\delta = \arcsin(0.3978 \sin(j' - 1.4 + 0.0355 \sin(j' - 0.0489))) \tag{8.1}$$

where δ is the Sun declination angle in radians and j' is expressed as day angle in radians:

$$j' = 360^\circ \cdot j/365.25, \quad j = 1, 2, ..., 365\,(366) \tag{8.2}$$

The position of the Sun in respect to a horizontal plane is defined by the solar altitude and solar azimuth. The solar altitude angle γ_s is a function of the solar hour angle ω, the solar declination angle δ and the latitude φ of the given location:

$$\sin \gamma_s = \cos \varphi \cos \delta \cos \omega + \sin \varphi \sin \delta \tag{8.3}$$

Given that the solar hour angle ω changes at 15° per hour and equals zero at noon it can be computed from local solar time t in decimal hours on the 24 hour clock:

$$\omega = 15^\circ \cdot (t - 12) \tag{8.4}$$

The solar azimuth angle α_s (a horizontal angle between the sun and meridian measured from east) can be expressed as a function of the solar altitude angle γ_s, the solar declination angle δ, and the latitude φ of the given location:

$$\cos \alpha_s = (\sin \varphi \sin \gamma_s - \sin \delta)/ \cos \varphi \cos \gamma_s \tag{8.5}$$

$$\sin \alpha_s = \cos \delta \sin \omega / \cos \gamma_s \tag{8.6}$$

The position of the Sun in respect to an inclined plane with slope β and aspect α (e.g., a hillslope or a building roof) is described by a solar incidence angle ν:

$$\cos \nu = \cos \gamma_s \sin \beta \cos(\alpha_s - \alpha) + \sin \gamma_s \cos \beta \tag{8.7}$$

To estimate the duration of solar radiation we compute the sunrise and sunset angle $\omega_{r,s}$ over a horizontal plane:

$$\cos \omega_{r,s} = -\tan \varphi \tan \delta \tag{8.8}$$

and the corresponding time of sunrise T_{sr} and sunset T_{ss} in hours is then:

$$T_{sr} = 12 - \omega_{r,s}/15^\circ \qquad T_{ss} = 12 + \omega_{r,s}/15^\circ \tag{8.9}$$

The sunrise/sunset angle over a south-facing inclined plane (for a location in the northern hemisphere) is computed as follows:

$$\cos\omega'_{r,s} = -\tan(\varphi - \beta)\tan\delta, \qquad \omega''_{r,s} = \min(\omega_{r,s}, \omega'_{r,s}) \tag{8.10}$$

where $\omega''_{r,s}$ is the sunrise and sunset angle over the inclined plane. More general equations can be found in Hofierka and Suri (2002).

Solar irradiation analysis has been implemented in GRASS GIS by Hofierka and Suri (2002) in the module *r.sun*. In addition to the direct solar irradiation described above the module incorporates diffuse and reflected irradiation and cast shadows. The module *r.sun* works in two modes. In the first mode it computes the solar incident angle in degrees and solar irradiance values in $\mathrm{W\,m^{-2}}$ for a specific instance in time. In the second mode it computes the daily sum of solar irradiation in $\mathrm{Wh\,m^{-2}\,day^{-1}}$ and the duration of the direct irradiation. To compute a series of direct irradiance maps for a given day with a selected time interval we use the add-on *r.sun.hourly*, which calls *r.sun* in a convenient loop. Similarly we use the add-on *r.sun.daily* to compute daily sums of direct irradiation for the entire year.

8.2 Case study: solar irradiation in urban environment

In this case study we studied how different spatial configuration of buildings changed direct solar radiation and cast shadows dynamics throughout the day. We placed wooden blocks of different shapes and sizes representing hypothetical buildings on a flat surface with an approximate scale of 1 : 300. We located this abstract site in Raleigh, North Carolina at latitude 36°. We examined two building configurations – one with four medium-sized houses with gabled roofs and the other with three medium-sized houses with gabled roofs and one tall building with a flat roof (Fig. 8.1a and 8.1b).

(a) (b) (c)

Fig. 8.1: Two configurations of buildings at approximately 1 : 300 scale viewed from the south. Fig. (c) shows the DSM and contours derived from a scan and projected over buildings.

8.2.1 The impact of building configuration on cast shadows

We compared the shadows cast by the buildings in both configurations during summer and winter solstice. After scanning the scenes we used the add-on *r.sun.hourly* to derive binary raster time series of cast shadows from 6:00 to 22:00 (local solar time) with a 30-minute time step. We had to specify the selected date as a day of year number. For example if we pick the summer solstice in 2015 we can easily compute the day of year using the Python *datetime* module:

```
import datetime
datetime.datetime(2015, 6, 21).timetuple().tm_yday
```

which gives us the number 172.

Running the following commands results in a time series of binary rasters with zeros representing cast shadow based on the direct (beam) radiation. The time series is registered in a spatio-temporal dataset and can be easily rendered as an animation and projected over the physical model.

```
r.slope.aspect elevation=scan aspect=aspect slope=slope
r.sun.hourly -t -b elevation=scan aspect=aspect slope=slope \
    start_time=6 end_time=22 time_step=0.5 day=172 year=2015 \
    beam_rad_basename=shadows_summer
g.gui.animation strds=shadows_summer
```

By changing the day parameter to 356 we obtained the time series of cast shadows for the winter solstice. Fig. 8.2 compares the cast shadows at 16:30 and 19:00 (local solar time) during the summer and winter solstice for the two different building configurations.

There is a clear difference in the length of cast shadows during winter and summer. The orientation of the cast shadows at the time close to sunset changes during the year as shown in Fig. 8.2a and 8.2c. During the summer the Sun sets below the horizon later (after 19:00) far to the northwest, while it sets earlier (after 16:30) and far to the southwest during winter. As a result the house in the southeast corner only has a view of the sunset during part of the year as the summer view is blocked by another building.

By adding a tall building to the scene we cast more shadow in the space between the houses and on houses in the east. Depending upon the season one of the houses on the east side becomes shadowed by the high building sooner than the other.

8.2.2 The impact of building configuration on direct solar irradiation

Cast shadows give us information about the availability of direct solar radiation at a particular time on particular days. However, we are often interested

Fig. 8.2: A comparison of cast shadows viewed from south. The first and second rows show two different building configurations. The first and second columns show cast shadows at 16:30 local solar time during the winter (a), (d) and summer solstice (b), (e), respectively. The third column shows cast shadows at 19:00 local solar time during the summer solstice. A figure for winter solstice at 19:00 is not shown as it would be after sunset.

in the cumulative solar irradiation, which tells us the amount of solar energy received on a given surface during a given time interval. Such information can then be used to assess the suitability of a location for solar energy applications.

To compute direct solar irradiation for individual days we called the add-on *r.sun.daily* with parameters similar to those we used for *r.sun.hourly*. We specified the first and last day of the year resulting in 365 daily irradiation maps. Additionally by specifying the `beam_rad` parameter we computed the sum of all of the daily maps. Again we can animate the time series projected over the scene to get more insight into the spatial patterns of solar radiation over the course of a year.

```
r.slope.aspect elevation=scan aspect=aspect slope=slope
r.sun.daily -t elevation=scan aspect=aspect slope=slope \
    start_day=1 end_day=365 day_step=1 \
    beam_rad=radiation_year_sum \
    beam_rad_basename=radiation_day_sum
g.gui.animation strds=radiation_day_sum
```

Fig. 8.3 shows direct solar irradiation in $\mathrm{Wh\,m^{-2}\,day^{-1}}$ during the winter solstice, spring equinox, and summer solstice with a common color ramp. The color ramp was specified in a plain text file `solar_color.txt` as:

```
100% red
70% yellow
0% gray
```

and then this color ramp was set for the entire time series:

```
t.rast.colors input=radiation_day_sum rules=solar_color.txt
```

(a) (b) (c)

Fig. 8.3: Direct solar irradiation in $\mathrm{Wh\,m^{-2}\,day^{-1}}$ during the winter solstice, spring equinox, and summer solstice

Fig. 8.4a shows the solar irradiation on a particular day in March in order to compare the cumulative solar irradiation for the two building configurations. In the first scenario there is variable irradiation in the space between the buildings as a result of the interaction between the spatial configuration of the buildings and the dynamic solar geometry. In the second scenario the pattern is even more pronounced because the street gets less direct sunlight. The slope of the buildings' roofs affects the amount of irradiation as a visual comparison of the color of the flat and tilted roofs clearly demonstrates. Furthermore, the shadow cast by the tall building has lowered the irradiation values on part of the roof of the southeastern house.

(a) (b)

Fig. 8.4: Solar irradiation patterns in March for different building configurations

References

Arnfield, A. (1990). Street design and urban canyon solar access. *Energy and Buildings*, 14(2):117–131.

Carrion, J. A., Estrella, A. E., Dols, F. A., Toro, M. Z., Rodríguez, M., and Ridao, A. R. (2008). Environmental decision-support systems for evaluating the carrying capacity of land areas: Optimal site selection for grid-connected photovoltaic power plants. *Renewable and Sustainable Energy Reviews*, 12(9):2358–2380.

Freitas, S., Catita, C., Redweik, P., and Brito, M. (2015). Modelling solar potential in the urban environment: State-of-the-art review. *Renewable and Sustainable Energy Reviews*, 41:915–931.

Hofierka, J. and Kaňuk, J. (2009). Assessment of photovoltaic potential in urban areas using open-source solar radiation tools. *Renewable Energy*, 34(10):2206–2214.

Hofierka, J. and Suri, M. (2002). The solar radiation model for Open source GIS: implementation and applications. In *Proceedings of the Open source GIS-GRASS users conference*, pages 1–19.

Jakubiec, J. A. and Reinhart, C. F. (2013). A method for predicting city-wide electricity gains from photovoltaic panels based on LiDAR and GIS data combined with hourly Daysim simulations. *Solar Energy*, 93:127–143.

Janke, J. R. (2010). Multicriteria GIS modeling of wind and solar farms in Colorado. *Renewable Energy*, 35(10):2228–2234.

Lobaccaro, G. and Frontini, F. (2014). Solar energy in urban environment: how urban densification affects existing buildings. *Energy Procedia*, 48:1559–1569.

Page, J., Albuisson, M., and Wald, L. (2001). The European Solar Radiation Atlas: a valuable digital tool. *Solar Energy*, 71(1):81 – 83.

Rigollier, C., Bauer, O., and Wald, L. (2000). On the clear sky model of the ESRA – European Solar Radiation Atlas – with respect to the heliosat method. *Solar Energy*, 68(1):33 – 48.

Scharmer, K. and Greif, J. (2000). *The European solar radiation atlas, vol. 2: database and exploitation software*. Presses del'Ecolle des Mines de Paris.

Chapter 9
Wildfire spread simulation

Forest fires, whether naturally occurring or prescribed, are potential risks for ecosystems and human settlements. These risks can be managed by monitoring the weather, prescribing fires to limit available fuel, and creating firebreaks. With computer simulations we can predict and explore how fires may spread. We can explore scenarios and test different fire management strategies under different weather conditions. Using Tangible Landscape and the GRASS GIS wildfire toolset we simulated several wildfire scenarios. We tested different configurations of firebreaks on the physical model and evaluated their effectiveness.

9.1 Fire spread modeling methods

Fire spread across landscapes is a complex, highly dynamic process influenced by weather, topography, and fuel. The fire spread models predict the rate of fire spread and fire perimeter growth using a combination of physical principles such as energy conservation and empirical parameters derived from observations and experimental data. For reviews of physical and empirical models of fire spread and methods of implementing fire simulation please refer to Sullivan (2009a,b,c).

9.1.1 Input data

The fuel model is one of the most important input variables for fire simulation. Albini (1976) and Anderson (1982) describe 13 classes of fuel (Table 9.1), which differ in fuel loads and the distribution of fuel particle size classes. The size of individual pieces of fuel influences the fire; heat is absorbed faster by small twigs due to their large surface to volume ratio. The moisture content of the fuel affects the amount of heat needed to ignite the fuel. The smaller the fuel size, the faster the fuel dries out and becomes more combustible.

Fuel class	Description
Grass and grass-dominated	
1	Short grass (1 foot)
2	Timber (grass and understory)
3	Tall grass (2.5 feet)
Chaparral and shrub fields	
4	Chaparral (6 feet)
5	Brush (2 feet)
6	Dormant brush, hardwood slash
7	Southern rough
Timber litter	
8	Closed timber litter
9	Hardwood litter
10	Timber (litter and understory)
Slash	
11	Light logging slash
12	Medium logging slash
13	Heavy logging slash

Table 9.1: The fuel model by Anderson (1982) describes 13 classes of fuel

The slope of the terrain and the speed and direction of the wind control the rate of spread and direction of the fire. Steeper slopes cause faster ignition in the upslope direction. This rule has been observed and quantified in several laboratory experiments (Viegas, 2004; Weise and Biging, 1996; Silvani et al., 2012).

Wind is specified by speed and direction at a given height. Unlike the slope of the terrain, wind is difficult to characterize due to its variability in space and time and spatially and temporally averaged values are used in the models. Sparks and embers can be carried great distances by strong winds; this process known as spotting allows fire to spread beyond firebreaks (Albini, 1983).

9.1.2 Fire spread algorithm

In our simulation we used the GRASS GIS modules *r.ros* and *r.spread*, implemented by Xu (1994), to compute the rate of spread and simulate the spread of fire.

The rate of spread computation is based on the BEHAVE model (Andrews, 1986). The inputs are the fuel model, fuel moisture, wind speed and

direction and terrain slope and aspect. The rate of spread is computed using the following equation:

$$R = \frac{I_R \xi (1 + \Phi_W + \Phi_S)}{\rho_b \varepsilon Q_{ig}} \tag{9.1}$$

where:

- R is the rate of spread in m/s
- I_R is the reaction intensity in kW/m
- ξ is the propagating flux ratio
- Φ_W is the wind coefficient
- Φ_S is the slope factor
- ρ_b is the oven-dry fuel per cubic meter of fuel bed in kg/m^3
- ε is the effective heating number
- Q_{ig} is the heat of preignition in kJ/kg.

For a detailed description of the underlying mathematical model and an explanation of the input data refer to Rothermel (1972).

The module *r.spread* uses Huygens' principle to simulate elliptically anisotropic spread where each cell center is a potential origin of spread and the local spread is ellipsoidal (Anderson et al., 1982). The sizes and orientations of the ellipses vary by cell. Local wind and slope directions determine the orientation of ellipses. The module *r.spread* uses a specific implementation of the shortest path algorithm.

We used the GRASS GIS add-on module *r.fire.spread* to call the modules *r.ros* and *r.spread* streamlining the fire spread computation and visualization. With this add-on we can use temporally variable input conditions and export the fire spread for each time step so that we can animate the simulation with the module *g.gui.animation*.

9.2 Case study: controlling fire with firebreaks

With Tangible Landscape we can intuitively create new fire management scenarios, rapidly testing how different sizes, shapes, and placements of firebreaks affect the spread of fire. To test different management strategies we simulated how a fire might spread across NCSU's Centennial Campus and tried to contain it by creating firebreaks of different sizes and alignments. In this case study we modeled a phenomenon – fire spread – that started outside the extent of the physical model, but spread towards the model (Fig. 9.1). Once the fire spread onto the model we could interact with it by creating firebreaks. Geospatial models and simulations need not be constrained to the boundary of the physical model, i.e. the extent of the tangible user interface. The processing extent can extend beyond the interactive area allowing us to work across scales.

We built a malleable physical model of the tree canopy by casting polymeric sand. To build the molds for casting we interpolated a DEM and a DSM from 2013 lidar data and then CNC routed the DEM and the inverse of the DSM. Then we cast a layer of sand representing the tree canopy between the two molds. Since this layer of sand represents tree canopy and thus fuel availability we could model clearcutting simply by removing sand. Thus we were able to intuitively design firebreaks by carving into the sand layer and reducing the fuel load. Please refer to Section 6.2.1 for a more detailed description of the site and model.

(a) (b)

Fig. 9.1: The study area on Centennial Campus: (a) an orthophotograph and (b) available fuel map projected over the table with the tangible 3D model.

9.2.1 Data preparation

To simulate fire spread we prepared several input raster layers for the module *r.ros* including the data about terrain, wind, fuel, and moisture conditions. We also specified the coordinates of location where the fire started.

Terrain Terrain slope influences the speed of spread and aspect influences the spread direction. After we set the region to our study area we derived the slope and aspect raster maps from the provided DEM:

```
g.region n=224134 s=222501 e=639326 w=637211 res=3
r.slope.aspect elevation=elevation slope=slope aspect=aspect
```

The DEM is also used as input for computing the maximum spotting distance.

Wind Wind has two components – midflame velocity and direction – and both can be spatially variable. In this case we used the prevailing wind speed and direction obtained from the nearest weather station through the State Climate Office of North Carolina.[1] The module *r.ros* requires velocity in feet

[1] State Climate Office of North Carolina: `www.nc-climate.ncsu.edu/`

per minute so the wind velocity data acquired from the State Climate Office of North Carolina must be converted from meters per second to feet per minute (to convert multiply by a factor of approximately 197). Wind direction is typically reported by the direction *from* which it originates, clockwise from the north. The module *r.ros*, however, requires the "to" direction (also clockwise from north). To simulate spatial variability we applied a random effect, for example, using the module *r.surf.gauss* to produce a raster map of Gaussian deviates with a specified mean and standard deviation:

```
r.surf.gauss output=wind_speed_avg mean=542 sigma=30
r.surf.gauss output=wind_dir mean=75 sigma=20
```

Fuel The module *r.ros* uses Anderson's 13 standard fire behavior fuel models. Fuel data layers at 30 m resolution for the USA are publicly accessible via the LANDFIRE website.[2] Most of our study site falls into fuel classes 8 and 9, i.e. timber litter (Fig. 9.1b).

Moisture Since fuel moisture data were not readily available for our site we generated the required, spatially variable raster maps of 1-hour fuel moisture and live fuel moisture percentage for dry conditions:

```
r.surf.gauss output=moisture_1h mean=10 sigma=5
r.surf.gauss output=moisture_live mean=20 sigma=5
```

Starting location The starting sources of the fire are represented as raster cells and can be created by digitizing points or importing coordinates from a file and converting them to a raster. We provided the fire starting location in a text file with the point coordinates (for example `638097,222934`) and converted them to a raster with the module *v.to.rast*:

```
v.in.ascii input=source.txt output=source separator=comma
v.to.rast input=source output=source type=point use=cat
```

9.2.2 Scenario with multiple firebreaks

In our first scenario the fire started near Trailwood Drive and was drawn to the northeast by a southwesterly wind. The fire jumped over the road and quickly spread towards the Chancellor's House (Fig. 9.2).

Using the following workflow we simulated and then animated the spread of the fire. The output raster maps generated by this workflow represent the relative time of ignition in minutes. To better visually represent the spread of the fire we used a red-yellow-gray color ramp. The steepness of the color gradient shows the relative rate of spread. The steeper the color gradient, the

[2] LANDFIRE: `http://www.landfire.gov/`

Fig. 9.2: The spread of the fire without intervention

slower the spread. To prepare the suggested color ramp we saved the color rules in a plain text file `fire_colors.txt`:

```
0% 50:50:50
60% yellow
100% red
```

To run the simulation we called the module *r.ros* once and then *r.spread* with the desired time lag parameter specifying the length of the simulation. To see the intermediate states we can run the module *r.spread* multiple times and assign our color ramp to the resulting spread raster map:

```
r.ros model=fuel moisture_1h=moisture_1h \
    moisture_live=moisture_live velocity=wind_speed_avg \
    direction=wind_dir slope=slope aspect=aspect \
    elevation=elevation base_ros=out_base_ros \
    max_ros=out_max_ros direction_ros=out_dir_ros \
    spotting_distance=out_spotting

r.spread -s base_ros=out_dir_ros max_ros=out_max_ros \
    direction_ros=out_dir_ros start=source \
    spotting_distance=out_spotting wind_speed=wind_speed_avg \
    fuel_moisture=moisture_1h output=spread_1 lag=20
r.spread -s -i base_ros=out_dir_ros max_ros=out_max_ros \
    direction_ros=out_dir_ros start=spread_1 \
```

```
    spotting_distance=out_spotting wind_speed=wind_speed_avg \
    fuel_moisture=moisture_1h output=spread_2 lag=20
r.spread ...

r.null map=spread_1 setnull=0
r.colors map=spread_1 rules=fire_colors.txt
```

Alternatively we can use the add-on modules *r.fire.spread*, which conveniently wraps the previous sequence of commands into a single command:

```
r.fire.spread -s start=source times=0 end_time=1600 \
    time_step=20 output=spread model=fuel \
    moisture_1h=moisture_1h moisture_live=moisture_live \
    direction=wind_dir slope=slope aspect=aspect \
    elevation=elevation speed=wind_speed_avg
```

After simulating the initial spread of the fire we attempted to prevent fire from spreading towards Chancellor's House by creating firebreaks. First we scanned the model to save the unmodified state. Next we manually removed sand (representing canopy) from the location where we want to have a firebreak (see Fig. 9.3a). We scanned the modified model and vertically matched the new scan to the unmodified scan using the function defined in code snippet 4.2.2. We derived the new fuel raster layer based on the difference between the two scans introducing no data values into the copy of the original fuel model using a simple raster algebra expression:

```
adjust_scan('scan_before', 'scan_after', 'scan_adjusted')
gscript.mapcalc("changed_fuel = if(scan_before - scan_adjusted
    > 0, null(), fuel)")
```

(a) (b)

Fig. 9.3: Creating a firebreak by (a) manually removing sand and then (b) scanning and detecting the change.

We repeated our simulation using the new fuel raster layer. The resulting fire spread in Fig. 9.4 shows that our attempt was only a partial success.

While we significantly slowed the spread of the fire, potentially giving firefighters more time to act, the fire jumped over the firebreak and started to spread towards Chancellor's House suggesting that a wider firebreak would be needed. With more precise data about fire behavior we could control and limit the spotting effect in the simulation.

Fig. 9.4: The spread of the fire after creating a firebreak

In our second scenario we added another firebreak to protect the neighborhood north of Lake Raleigh. The lake creates a natural barrier that constrains the spread of the fire. To protect the community to the north we extended this barrier by creating an additional firebreak starting on western shore of the lake (Fig. 9.5a). After we rescanned the model and reran the simulation, we found that the additional firebreak was only locally effective – while the fire was significantly slowed, it would have eventually reached the neighborhood.

(a) (b)

Fig. 9.5: The simulation with additional firebreak: (a) the creation of a new firebreak by removing sand and (b) the result of the new simulation.

References

Albini, F. A. (1976). Estimating wildfire behavior and effects. Technical report, Intermountain Forest and Range Experiment Station, Forest Service, U.S. Dept. of Agriculture.

Albini, F. A. (1983). Potential spotting distance from wind-driven surface fires. Technical report, Intermountain Forest and Range Experiment Station, Forest Service, U.S. Dept. of Agriculture.

Anderson, D. H., Catchpole, E. A., De Mestre, N. J., and Parkes, T. (1982). Modelling the spread of grass fires. *The Journal of the Australian Mathematical Society. Series B. Applied Mathematics*, 23:451–466.

Anderson, H. E. (1982). Aids to determining fuel models for estimating fire behavior. Technical report, Intermountain Forest and Range Experiment Station, Forest Service, U.S. Dept. of Agriculture.

Andrews, P. L. (1986). BEHAVE: fire behavior prediction and fuel modeling system – BURN subsystem, Part 1. page 130.

Rothermel, R. C. (1972). A mathematical model for predicting fire spread in wildland fuels. Technical report, Intermountain Forest and Range Experiment Station, Forest Service, U.S. Dept. of Agriculture.

Silvani, X., Morandini, F., and Dupuy, J.-L. (2012). Effects of slope on fire spread observed through video images and multiple-point thermal measurements. *Experimental Thermal and Fluid Science*, 41:99 – 111.

Sullivan, A. L. (2009a). Wildland surface fire spread modelling, 1990–2007. 1: Physical and quasi-physical models. *International Journal of Wildland Fire*, 18(4):349–368.

Sullivan, A. L. (2009b). Wildland surface fire spread modelling, 1990–2007. 2: Empirical and quasi-empirical models. *International Journal of Wildland Fire*, 18(4):369–386.

Sullivan, A. L. (2009c). Wildland surface fire spread modelling, 1990–2007. 3: Simulation and mathematical analogue models. *International Journal of Wildland Fire*, 18(4):387–403.

Viegas, D. X. (2004). Slope and wind effects on fire propagation. *International Journal of Wildland Fire*, 13(2):143–156.

Weise, D. R. and Biging, G. S. (1996). Effects of wind velocity and slope on flame properties. *Canadian Journal of Forest Research*, 26(10):1849–1858.

Xu, J. (1994). *Simulating the spread of wildfires using a geographic information system and remote sensing*. Phd dissertation, Rutgers University, New Brunswick.

Chapter 10
Coastal modeling

In this chapter we discuss how to simulate inundation and flooding in coastal landscapes, explain workflows for developing storm surge and sea level rise scenarios, and present two case studies. The first case study is a series of storm surge and dune breach scenarios for a populated barrier island. The second is a design – informed by flood and erosion modeling – for resilient, sustainable guest housing at a coastal research institute.

10.1 Modeling potential inundation

Inundation and flooding in coastal landscapes is caused by storm surge and sea level rise. It is very complex and depends upon topography and bathymetry, tides, sea level surge and waves, and wind and rainfall. The spatial extent of flooding can be approximated by "spreading" the observed or predicted water level throughout the landscape (Clinch et al., 2012).

To approximate inundation and flooding we can spread the given water level from a predefined seed located in the ocean using the module *r.lake*. This approach respects the obstacles, so if a foredune protects a low lying area, this area will not be flooded even if the water level exceeds its elevation. We describe the algorithm as follows: an increase in water level approximates the dynamics of inundation, but we assume that the water spreads at an infinite speed and thus floods everything that it can reach instantly. Therefore this method is best used to map the maximum possible extent assuming that there are no changes in topography during the process.

10.2 Case study: simulating flood risks for barrier island communities

We explored the impact of topography on inundation and flooding at the Jockey's Ridge sand dune complex in Nags Head on the Outer Banks barrier

Fig. 10.1: The Jockey's Ridge dune complex in Nags Head, North Carolina

islands of North Carolina (Fig. 10.1) with a series of a storm surge scenarios. We carved breaches into the foredune of our sand model to identify which homes were most vulnerable to storms.

For this study we selected a study area including the beach, foredune, homes, road, dune complex, maritime forest, and soundside shore. We derived a bare earth DEM and a DSM with buildings and vegetation for our study from a 2009 lidar survey. Then we cast a polymeric sand model of the DEM from a CNC routed mold. We projected an orthophotograph, elevation data, the shoreline, and building footprints as context. Based on the building footprints we added massing models of the buildings to our sand model.

To study the potential impacts of a hurricane we simulated storm surge at increasing levels from 1 m to 4 m using the add-on module *r.lake.series*. Then we breached the foredune at a point where it has been compromised by the construction of boardwalk for beach access and compared the extent of flooding at different surge levels (Fig. 10.2).

```
r.lake.series elevation=elev_2009_1m output=flood \
    start_water_level=1 end_water_level=4 water_level_step=0.1 \
    coordinates=913525,249507
g.gui.animation strds=flood
```

(a) (b)

Fig. 10.2: Inundation extent simulations for Jockey's Ridge area for (a) 1 m and (b) 2 m of storm surge.

10.3 Case study: designing resilient coastal architecture

(a) A rendering of the proposed design for a residential pavilion

(b) A masterplan of the UNC Coastal Studies Institute with the proposed residential pavilions

Fig. 10.3: The students' design for a series of pavilions for the UNC Coastal Studies Institute on Roanoke Island, NC. Images by David Koontz and Faustine Pastor.

Two architecture students – David Koontz and Faustine Pastor – used Tangible Landscape to site a series of residential pavilions (Fig. 10.3) for visitors staying at the UNC Coastal Studies Institute on Roanoke Island, North

Carolina as part of an joint architecture-engineering studio. They used storm surge and water flow simulations to inform the selection of building sites. Then they spaced pilings for the buildings and adaptively graded a biofiltration swale based on near real-time feedback about water flow, sediment flux, and erosion-deposition.[1]

The UNC Coastal Studies Institute – the site for the studio – is a research institute on Roanoke Island in the Outer Banks of North Carolina. Students in this North Carolina State University studio course led by Professors Joe DeCarolis, David Hill, and Ranji Ranjithan were tasked with designing sustainable housing for guests and researchers visiting the institute.[2] The site is at risk of inundation from sea level rise and storm surge and is in a sensitive back-barrier salt marsh.

Given the risk of flooding David and Faustine used Tangible Landscape to visualize different levels of inundation and test the vulnerability of potential building sites. They placed basswood massing models of buildings on a cast sand model of the landscape and then used the module *r.lake* to simulate 4 m storm surge and check if their buildings would stay dry (Fig. 10.4). In their design the buildings would be raised on piers and connected by boardwalks over a bioswale. They simulated water flow to find the right spacing for the piers over swale. Then they adaptively graded the swale using feedback from water flow, erosion-deposition, and sediment flux simulations, sculpting the channel so that water and sediment flowed towards a bioretention pond (Fig. 10.5).

[1] Watch the video on Youtube: `https://youtu.be/PbbzWymGvLo`

[2] Read more about the studio on the Coastal Dynamics Design Lab website: `http://design.ncsu.edu/coastal-dynamics-design-lab/`

(a) (b) (c)

Fig. 10.4: Students using water flow and flood simulations to adaptively inform the placement of a series of pavilions for the Coastal Studies Institute on Roanoke Island, NC: (a) siting buildings by hand on an artificial ridge, (b) studying the flow of water around the buildings and (c) testing whether the buildings stay dry with 4 m storm surge.

(a) (b) (c)

Fig. 10.5: Students using (a) water flow, (b) erosion-deposition, and (c) sediment flux simulations to space structural piers and adaptively grade a biofiltration swale for the Coastal Studies Institute on Roanoke Island, NC.

References

Clinch, A. S., Russ, E. R., Oliver, R. C., Mitasova, H., and Overton, M. F. (2012). Hurricane Irene and the Pea Island breach: prestorm site characterization and storm surge estimation using geospatial technologies. *Shore & Beach*, 80(2):38–46.

Appendix A

A.1 Applications of Tangible Landscape

In this section we briefly describe additional applications of Tangible Landscape that we, our students, and our colleagues have developed. For each project we explain the aim, used methods and studied processes.

Modeling avalanches in High Tatras

Eva Stopkova modeled snow avalanches in the High Tatras mountain range in Slovakia (Stopková, 2007, 2008). Avalanche potential is closely related to landscape topography so Stopkova built a physical model of her study area in High Tatras in order to visualize the spatial pattern of topographic parameters and landforms (Fig. A.1). She was able to project the results of avalanche danger assessment over the model, but she was not able usefully compute the assessment using the modified and scanned physical model. The scanned models were not accurate enough for practical applications due to their small scale and the steepness of their slopes. Modeling avalanches with Tangible Landscape would require a larger physical model at a larger map scale. CNC routing can be used to precisely and efficiently fabricate the size of model needed for such an application.

Visualizing the evolution of Oregon Inlet

Liliana Velasquez Montoya created a projection augmented sand model of Oregon Inlet where she was modeling how storms drive the morphological evolution of barrier islands. To visualize and present the results of her research Montoya projected a video depicting the evolution of the inlet in response to storm events onto a sand model. She built the sand model to physically represent the present 3D form of the inlet. Since the sand model was not dynamic it only matched the end of animation. This was a useful mode of

Fig. A.1: A sand model of High Tatras in Slovakia augmented with a projection of the landforms computed by the module *r.geomorphon*; a legend and a diagram of the landform types are projected on the table beside the model.

representation as the model acted as a datum that helped us compare the present and past morphology of the inlet. Since the small sand model was only large enough to show the most active part of the inlet she also projected the surrounding coastal landscape (Fig. A.2).

Fig. A.2: A sand model of Oregon Inlet with a projected video showing its formation and evolution along with a graph of wave height changing over time.

Designing disaster relief housing for Rodanthe

Logan Free designed disaster relief shelters for an emergency ferry terminal at Rodanthe on the Outer Banks of North Carolina. The ferry terminal is used for evacuation and relief when there are disasters like hurricanes or bridge failures. Free was tasked with designing temporary housing structures and a community service center for disaster recovery at the ferry terminal. To identify vulnerable hotspots and select safe sites for the shelters he built a

sand model of the site and used Tangible Landscape with the module *r.lake* to simulate the potential extent of storm surge flooding (Fig. A.3).

(a) (b)

Fig. A.3: Simulating storm surge at the Rodanthe emergency ferry terminal on the Outer Banks of North Carolina: (a) the mean high water level on the sound side of the island and (b) the approximate water level during high storm surge.

Simulating landscape change in Charlotte

Douglas A. Shoemaker created a model of a small watershed in Charlotte, North Carolina to test how the FUTure Urban-Regional Environment Simulation (FUTURES) framework could be integrated with Tangible Landscape. FUTURES uses a stochastic patch growing algorithm to simulate urban growth and land use change (Meentemeyer et al., 2013). It has been integrated with GRASS GIS as the add-on module *r.futures* (Petrasova et al., 2015). Shoemaker used the raster digitizer tool in the GRASS GIS GUI to designate a site for development and change the land cover. He also sculpted the sand to grade the topography for the new development. Then he ran FUTURES to explore how the new development would influence longterm landscape change in the watershed-scale study area (Fig. A.4).

Reconstructing a Paleolake in Mongolia

Gantulga Bayasgalan used a sand model to reconstruct a paleolake in the Galuut Valley in central southwest Mongolia. He modeled alternative reconstructions of the landscape in sand and used Tangible Landscape to compute the flow accumulation pattern, geomorphological features, and several other characteristics for each landscape configuration (Fig. A.5a).

Cell tower planning in Athens county

Jesse Boyd used Tangible Landscape to site cell towers in Athens county, Ohio. To find good sites for cell towers, Boyd tested different spatial configurations by placing markers representing cell towers on a physical model of

(a) (b)

Fig. A.4: Landscape change modeling for a watershed in Charlotte, North Carolina: (a) a sand model of the terrain at the Charlotte study area augmented with a projected hillshade to enhance perception of relief; (b) using a mouse and the digitizer tool to draw a polygon to designate a development area.

(a) A reconstruction of a paleolake in the Galuut Valley, Mongolia showing the landforms computed by *r.geomorphon*

(b) A physical model used to site cell towers in Athens county, Ohio

Fig. A.5: Reconstructing a paleolake and a terrain model used for cell tower planning

the landscape (Fig. A.5b). Since topography blocks cellular signals he used viewshed analysis to estimate cellular coverage. He experimented with different cell tower heights by changing the observer elevation parameter of the module *r.viewshed*.

Monitoring coastal erosion

Tristan Dyer used terrestrial lidar data to monitor coastal erosion on a beach near the US Army Corps of Engineers' Field Research Facility in Duck on the Outer Banks of North Carolina. Coastal erosion can be visually and tactilely explored by building a sand model for each lidar survey. Dyer scanned a transect of the beach with a terrestrial lidar system, interpolated the lidar point cloud as a DEM, and then built a sand model with the aid of Tangible Landscape to quickly prototype and visualize the landform (Fig. A.6a). A sequence of these models would show the morphological evolution of the beach.

Exploring impacts of a beaver dam

Karl W. Wegmann and his students used Tangible Landscape to study how beaver dams affect streams in North Carolina. They used Tangible Landscape to model the morphology and flow of a stream with and without beaver dams. First they built a sand model of a segment of the existing stream with beaver dams and studied its topographical parameters and hydrological processes. Then they sculpted the sand model and computed different water flow scenarios to study how the stream could have been without beaver dams (Fig. A.6b).

In this application we merged the scanned elevation model with a DEM of the surrounding landscape to put the modeled area into its broader context and account for water flowing from the entire contributing watershed. The water flow simulation was then run over the entire area so that the water from the contributing watersheds would fill the stream.

(a) A sand model of a beach in Duck, North Carolina used to study beach erosion

(b) A sand model of a valley in North Carolina Piedmont used to study how beaver dams influence stream flow and morphology

Fig. A.6: Sand models used to study beaches and streams

Modeling the potential impacts of a coal ash pond spill

Matthew Horvath (2014) studied the potential impacts of spills from the Cape Fear Plant coal ash ponds near Moncure, North Carolina. Horvath built a sand model for a study area near the Cape Fear river and used Tangible Landscape with the add-on module *r.damflood* to simulate a breach in a coal ash pond (Fig. A.7). Artificial landforms kept the simulated spread of contaminated surface water from reaching the nearby Cape Fear river.

Testing a landform migration algorithm

Petras et al. (2015) used Tangible Landscape to test the behavior of a newly developed algorithm for describing landform migration. The algorithm maps

(a) (b)

Fig. A.7: Making a breach in a coal ash pond (a) and simulating the spread of contaminated water (b).

the horizontal migration of complex landforms as gradient fields based on the analysis of contour evolution. The gradient field is a vector field representing the movement and deformation of contours. It can be used to quantify the rate and direction of landform migration at any point in space and time. Data for testing algorithms is usually computer generated. In this case, however we wanted to create simplified, yet realistic abstractions of landforms that would allow us to clearly model and visualize landform migration. It was easier to make these abstract landforms by hand than on the computer. To create test data in a controlled environment we built sand models by hand to represent a moving landform. Using Tangible Landscape we sculpted and scanned a model of the initial terrain and then sculpted and scanned a model of the terrain after it had migrated to create an elevation time-series (Fig. A.8).

(a) (b)

Fig. A.8: Iteratively sculpting a landform in sand to generate time-series data: (a) the initial state followed by (b) changing the position and shape of the landform.

Managing the spread of Sudden Oak Death in Sonoma Valley

In collaboration with Francesco Tonini and Douglas A. Shoemaker we used Tangible Landscape to test the effectiveness of different treatment scenarios for controlling the spread of Sudden Oak Death in Sonoma Valley, California. We used a stochastic model to simulate the spread of Sudden Oak Death and Tangible Landscape to visualize the spread and designate new treatment areas. We CNC routed a large (1 m $\times$ 1 m) MDF model of the terrain and coated it with magnetic paint. To make markers that would hold in place on the complex topography we attached adhesive magnetic strips to the base of basswood markers. We used the markers to collaboratively draw polygons representing treatment areas with Tangible Landscape (Fig. A.9a). The polygons reduced the host species of the disease on aligned pixels thus influencing the next run of the simulation. Each treatment area was automatically assigned a label with its area. The sum of all of the treatment areas was dynamically displayed on a simple dashboard. The disease spread simulation was written in the R (**?**) statistical programing language and environment.

(a) (b)

Fig. A.9: Using object recognition to digitize polygons and pixels representing treatment areas in order to computationally steer (a) Sudden Oak Death in Sonoma Valley, California, and (b) termites spread simulation in Fort Lauderdale, Florida.

Managing the spread of termites in Fort Lauderdale

In collaboration with Francesco Tonini we used Tangible Landscape as a spatial decision support tool for collaboratively managing the spread of termites in Dania Beach, Fort Lauderdale, Florida. We used pixel-based object recognition to interact with a stochastic simulation of the spread of termites. Placing a marker on a flat surface with a projected grid changed a value on

the corresponding pixel in the simulation input – a raster index of termite habitat (Fig. A.9b).

Tangible exploration of subsurface data

Using Tangible Landscape we explored the 3D distribution of the percentage of subsurface moisture (Petrasova et al., 2014) measured in Kinston, North Carolina and represented as a GRASS GIS 3D raster (GRASS Development Team, 2015). By digging into the sand model with our hands we could explore underground as if we were at an excavation site. As we dug into the sand the cross-section of the scanned surface with the 3D raster representing soil properties was projected in real-time (Fig. A.10). We also projected additional GIS layers (such as an orthophotograph, elevation contours, and flow accumulation) over the model to provide spatial context and gain further insights into soil moisture distribution and its relation to the landscape.

(a) (b)

Fig. A.10: Exploring subsurface moisture by removing sand and projecting the cross-section of a scanned surface with the 3D raster of moisture.

Additionally, in combination with marker detection from Section 6.2.2 we can create and visualize vertical profiles or soil core samples. This method is an intuitive and natural way of exploring subsurface data and it represents an alternative to more abstract 3D computer visualization tools.

References

GRASS Development Team. 3D raster data in GRASS GIS [online]. (2015) [Accessed 2015-08-13]. `http://grass.osgeo.org/grass70/manuals/raster3dintro.html`.

Horvath, M. (2014). Spill impacts from coal ash pond using GRASS GIS. Upublished poster.

Meentemeyer, R. K., Tang, W., Dorning, M. A., Vogler, J. B., Cunniffe, N. J., and Shoemaker, D. A. (2013). FUTURES: multilevel simulations of emerging urban-rural landscape structure using a stochastic patch-growing algorithm. *Annals of the Association of American Geographers*, 103(4):785–807.

Petras, V., Mitasova, H., and Petrasova, A. (2015). *Mapping gradient fields of landform migration*, pages 173 – 176. Bogucki Wydawnictwo Naukowe, Adam Mickiewicz University in Poznań - Institute of Geoecology and Geoinformation, Poznań, Poland.

Petrasova, A., Harmon, B., Mitasova, H., and White, J. (2014). Tangible Exploration of Subsurface Data. Poster presented at 2014 Fall Meeting, AGU, San Francisco, Calif., 15-19 Dec.

Petrasova, A., Petras, V., Shoemaker, D. A., Dorning, M. A., and K. Meentemeyer, R. (2015). The integration of land change modeling framework FUTURES into GRASS GIS 7. In *Geomatics Workbooks n 12 – "FOSS4G Europe Como 2015"*.

Stopková, E. (2007). Modelovanie oblastí vzniku lavín s využitím GIS. In *Sborník studentské konference GISáček*.

Stopková, E. (2008). Predikcia lavínového nebezpečenstva s využitím GIS. Bachelor's thesis, Slovak University of Technology in Bratislava.

A.2 Data sources

Throughout the book we use various data sources. Here is a list of the most useful data sources for the study areas presented in the book.

Sample data for this book

NCSU OSGeo Research and Education Laboratory

`http://geospatial.ncsu.edu/osgeorel/data.html`

US lidar data

United States Interagency Elevation Inventory

`http://coast.noaa.gov/inventory/`

Earth Explorer

`http://earthexplorer.usgs.gov/`

Digital Coast

`http://coast.noaa.gov/dataviewer/`

Open Topography

`http://www.opentopography.org/`

US digital elevation models

National Elevation Dataset on the National Map Viewer

http://viewer.nationalmap.gov/viewer/

US orthoimagery

USGS EROS Orthoimagery web mapping service

http://isse.cr.usgs.gov/arcgis/services/Orthoimagery/USGS_EROS_Ortho_1Foot/ImageServer/WMSServer

US soil surveys

USDA Web Soil Survey

http://websoilsurvey.sc.egov.usda.gov

US fire modeling data

LANDFIRE

http://www.landfire.gov/

Global datasets

GRASS GIS community list of global datasets

http://grasswiki.osgeo.org/wiki/Global_datasets

NC climate data

State Climate Office of North Carolina

http://www.nc-climate.ncsu.edu/

A.3 Starting with GRASS GIS

To run examples from this book we provide a dataset which can be downloaded from the NCSU OSGeo Research and Education Laboratory website.[1] Once the downloaded dataset is unzipped we create a directory `grassdata`

[1] http://geospatial.ncsu.edu/osgeorel/data.html

in our home directory and place the dataset there. Then we start GRASS GIS and a start up dialog in Fig. A.11 appears on screen. In the upper text field we specify the path to our `grassdata` directory. Then we should be able to see Location `nc_spm_tl` containing several Mapsets. We select mapset `practice1` and press the Start button.

Fig. A.11: Starting GRASS GIS

After opening the Layer Manager and Map Display windows we display the raster map `elevation` as shown in Fig. A.12. To execute commands from this book presented in boxes with green background we paste the commands into the *Command console* tab as shown in Fig. A.13a and press Enter. The Python code in boxes with blue background can be executed in the *Python shell* tab (Fig. A.13b) or in any Python shell running in a GRASS GIS session.

More instructions how to use GRASS GIS GUI are available on the official GRASS GIS introduction page[2] and in the GRASS GIS GUI manual.[3]

[2] `http://grass.osgeo.org/grass70/manuals/helptext.html`

[3] `http://grass.osgeo.org/grass70/manuals/wxGUI.html`

Fig. A.12: Display raster map elevation: click on *Add raster map layer* button in toolbar, select `elevation` from drop-down list, and press OK.

(a) (b)

Fig. A.13: GRASS GIS modules can be run from (a) the *Command console* tab or (b) Python shell using Python syntax.