UNIVERSITY OF CALGARY

University of Calgary

PRISM: University of Calgary's Digital Repository

Graduate Studies | The Vault: Electronic Theses and Dissertations

2018-05-23

Landscaper: A Modeling System for 3D Printing Scale Models of Landscapes

Haji Allahverdi Pour, Kamyar

Haji Allahverdi Pour, K. (2018). Landscaper: A Modeling System for 3D Printing Scale Models of Landscapes (Unpublished master's thesis). University of Calgary, Calgary, AB.
doi:10.11575/PRISM/31943
http://hdl.handle.net/1880/106676
master thesis

University of Calgary graduate students retain copyright ownership and moral rights for their thesis. You may use this material in any way that is permitted by the Copyright Act or through licensing that has been assigned to the document. For uses that are not allowable under copyright legislation or licensing, you are required to seek permission.

Downloaded from PRISM: https://prism.ucalgary.ca

UNIVERSITY OF CALGARY

Landscaper: A Modeling System for 3D Printing Scale Models of Landscapes

by

Kamyar Haji Allahverdi Pour

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

GRADUATE PROGRAM IN COMPUTER SCIENCE

CALGARY, ALBERTA

MAY, 2018

© Kamyar Haji Allahverdi Pour 2018

Abstract

Landscape models of geospatial regions provide an intuitive mechanism for exploring complex geospatial information. However, the methods currently used to create these scale models require a large amount of resources, which restricts the availability of these models to a limited number of popular public places, such as museums and airports. In this thesis, we have proposed a system for creating these physical models using an affordable 3D printer in order to make the creation of these models more widely accessible. Our system retrieves GIS data relevant to creating a physical model of a geospatial region and then addresses the two major limitations of affordable 3D printers, namely the limited number of materials and available printing volume. This is accomplished by separating features into distinct extruded layers and splitting large models into smaller pieces, allowing us to employ different methods for the visualization of different geospatial features, like vegetation and residential areas, in a 3D printing context. We confirm the functionality of our system by printing two large physical models of relatively complex landscape regions.

Acknowledgements

I would like to thank all the people that helped me to accomplish my degree. First of all, I am grateful to my supervisor Dr. Samavati for his support, knowledge and understanding. During my studies, I learned a great deal from him and I really appreciate all the time he put into helping me for my research.

I want to thank Ali for his great insights that helped me progress my research in a good direction. I would also like to thank Hessam for his help that greatly improved my research results.

Thanks to my fellow researchers, Troy, Mark, Tim, Amirhessam, Hasan and Samin for their comments and support during my research. Special thanks to Troy for spending a lot of time reviewing my paper and my thesis and giving me a great amount of helpful comments.

I would like to thank my family for their support and encouragement that made my studies possible. I also want to thank my dear wife, Saharnaz, for her love and support during my studies.

Table of Contents

List of Figures and Illustrations

List of Symbols, Abbreviations and Nomenclature

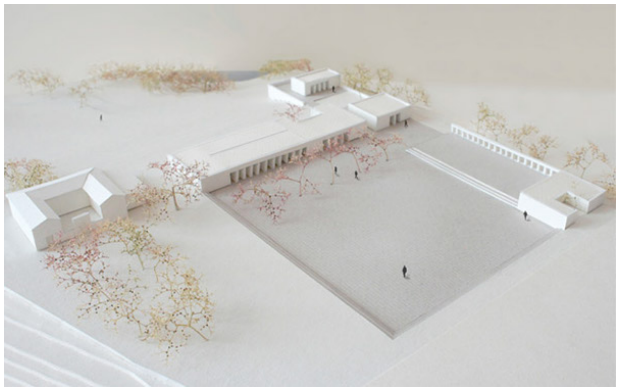
Symbol	Definition
GIS	Geographic Information System
GPS	Global Positioning System
UAV	Unmanned Aerial Vehicle
DSLR	Digital Single-Lens Reflex
CNC	Computer Numerical Control
RP	Rapid Prototyping
AM	Additive Manufacturing
FDM	Fused Deposition Modeling
SLA	Stereolithography
LiDAR	Light Detection and Ranging
DEM	Digital Elevation Model
CDSM	Canadian Digital Surface Model
CDT	Constrained Delaunay Triangulation
OSM	OpenStreetMap
PIC	Polygon Interior Covering
CGAL	The Computational Geometry Algorithms Library
JSON	Javascript Object Notation
PLA	Polylactic Acid
MPFR	Multiple Precision Floating-Point Reliably

Chapter 1

Introduction

Visualizing geospatial features is a challenging task. One contributing factor to this challenge is the variety and size of geospatial data sets. Another lies in how to assign the data to an appropriate representation of the Earth for a better understanding of regions and their geospatial features. For example, how to visualize hiking trails in a national park or ski trails in mountains such that people can better imagine the place and its geospatial features. Physical visualization is beneficial as a 3D physical model enables tactile exploration and easy circumnavigation [Hull and Willett, 2017]. These attributes of 3D physical models (i.e. scale models) help to better explore complex GIS data, which makes this type of physical visualization useful for both educational and scientific purposes. Moreover, urban designers, urban planners and architects can benefit from these scale models when presenting their 3D designs to non-experts [Hull and Willett, 2017]. Currently, popular public places such as national parks, airports and ski resorts benefit from these models as they help visitors develop a familiarity with the area quickly. For example, in Figure 1.1e, a scale model of Mount Rainier National Park, available in Longmire wilderness information center, helps visitors get familiar with the trails in this national park. Also, this type of physical visualization can be employed to raise awareness about environmental issues, such as global warming, by highlighting impacts to endangered regions of interest over time. A number of scale models can be seen in Figure 1.1.

The most common method used to create such scale models is to use subtractive machining with a CNC (Computer Numerical Control) machine, where the required model is machined out of a sheet or volumetric material. Laser cutting, a 2D CNC machining method, is considered the most affordable option when using CNC. Using laser cutting, sheets of ma-



(a) Photo by Marion Boissières (CC BY-NC 4.0).

(b) Photo by Phil Parker (CC BY 2.0).

(c) Scale model of Topkapi Palace in Istanbul.

(d) Photo by Hsiu-i Lee (CC BY-NC-ND 4.0)

(e) Photo by Joe Mabel (CC BY-SA 3.0)

(f) Photo by Jordiferrer (CC BY-SA 3.0)

Figure 1.1: Several examples of scale models.

terial are cut and later stacked to create a 3D model.

Rapid Prototyping (RP) is another method that uses Additive Manufacturing (AM), where models are created by adding layers of material. Stereolithography (SLA) and Fused Deposition Modeling (FDM) are two RP-based approaches that can operate on affordable desktop 3D printers. SLA uses an ultraviolet laser to solidify a rather costly liquid material, such as resin, while FDM operates by melting and extruding a plastic material.

Every 3D printing process imposes several difficulties on the creation of scale models. CNC is limited to only one material per session and requires costly machines. Methods using RP are generally considered inappropriate for building large landscape models due to the size limitation of machines that use this method [Kvan et al., 2001]. Specifically, FDM suffers from a lower accuracy, problems with overhanging parts, warpage and a limited number of materials that may be used per session.

1.1 Goals

The main research goal of this thesis is to design a system that uses affordable 3D printers (i.e. FDM based) to create scale models of geospatial landscapes. We target the general scale of landscape models, which is between 1:1000 and 1:2500, for creating these scale models. While SLA is a reasonable method to use with our system, as it is currently a more expensive option, we utilize FDM due to its affordability and availability. Using our system we can create 3D models that both respect the GIS data and the FDM process limitations. We propose methods to 3D print large physical models using such 3D printers, while providing methods to physically visualize complex geospatial features such as vegetation and narrow roads without the creation of overhangs or highly delicate pieces. Furthermore, we consider the accuracy limitation of these 3D printers in both the design of these 3D models and their assembly.

1.2 Problems and Challenges

Since geospatial areas are usually complex with large fine features (e.g. valleys, trails, roads and vegetation), creating a 3D model of these regions that is suitable for 3D printing is not an easy task. Although one may assume the use of highly detailed data sets (e.g. LiDAR), these kinds of data sets are only available for small regions, and capturing them requires a substantial amount of resources. Furthermore, landscape models are generally created at scales between 1:1000 and 1:2500. At these scales, utilizing the highly detailed information of a region of interest is neither necessary nor sufficient for 3D printing. For example, a perfect tree model has many small delicate features at these scales and is not tailored for 3D printing. On the other hand, vast numbers of 2D and 2.5D geographical data sets (i.e. digital elevation models (DEM), satellite images and GIS vector data) are freely available. These data sets have been captured for most areas of the globe through the activities of various providers and governmental initiatives (e.g. OpenStreetMap, CDSM [CDS, 2014]). Therefore, a core challenge to creating a physical model of any region is the question of how to produce a detailed 3D model out of these freely available yet less-detailed datasets. Moreover, we must also address how such models can be designed to respect the limitations of the 3D printer. For example, there are challenges in segmenting large 3D models into small, yet robust, printable pieces and in incorporating different colors into the final result. Additional challenges include the printing of complex features in a landscape model, such as vegetation and residential areas, and assisting the assembly of the physical model.

1.3 Methodology

We introduce a modeling system to address the challenges. The inputs to our system are DEM and 2D GIS vector data. From these data sets our system generates a series of 3D meshes, each of which represents a unique geospatial feature from the input data. These feature meshes are further processed to create a collection of 3D elements, where each element

is suitable for printing. After printing these 3D elements, they are assembled using a manual process to create the final physical model.

To create 3D feature meshes, we employ Constrained Delaunay Triangulation (CDT) to generate a 2D triangulation that respects all of the 2D input features. Using the input DEM, this 2D triangulation is turned into a 3D mesh. A region growing algorithm is applied to this 3D mesh to extract separate feature submeshes for each input feature. As we use affordable 3D printers, there is a printable size limitation. A grid optimization is used to break large feature submeshes into printable 3D elements.

In order to support multiple colors and materials in FDM printers, we create multiple extruded layers. In this way, a lake can be printed in blue and a vegetation region can be printed in green as part of another printing session. However, a forest region and a grass field are hard to differentiate without additional information, as they have similar green colors. To address this issue, while considering the printable models, we create appropriate designs and patterns for different features to assist with feature recognition.

To improve the manual assembly process, our system generates 3D models with a proper coding based on the feature number and grid location, resulting from the segmentation algorithm. Our system also avoids creating very small pieces that are harder to assemble, by using a cost function during segmentation. In regions whose features have complex interactions (e.g. residential areas), our system creates stencil layers that ease the attachment of features. To help with the fitting of pieces, our system has a configurable offset amount to be applied to each piece based on the specific 3D printer inaccuracy. To further enhance this manual process, our system packs smaller pieces together to reduce the number of printing sessions required. We also help reduce printing time by creating pillars to support the surfaces of the physical models.

1.4 Contributions

Our main contribution is to introduce and develop a system that, given a geographical region and its associated GIS data, creates 3D elements of a scale model that are printable by affordable 3D printers, e.g. FDM based printers. We have proposed methods to physically visualize geospatial features that are generally difficult to 3D print for affordable 3D printers. By employing displacement maps for vegetation regions and stencil layers for residential areas, our system speeds up the assembly process for massive amounts of models, while respecting the input geospatial data sets. By decomposing a landscape region into its features, we provide a way to 3D print a scale model using 3D printers that support only a limited number of materials. Our system provides a local grid segmentation algorithm to create smaller pieces that fit inside the printable volume of 3D printers. This segmentation operates on every feature mesh locally, which makes the algorithm more flexible than a global grid for the whole physical model. Furthermore, since the virtual model of a scale model is rather highly detailed and complex, separating features and segmenting them helps in creating smaller 3D elements that are more manageable for the 3D printing software to process. Lastly, the 3D elements created by our system can be assembled easily after printing to create large landscapes at scales between 1:1000 and 1:2500. The main contribution of this thesis has been accepted by EuroVis 2018 and will appear as a full paper in a special issue of Computer Graphics Forum (CGF) [Allahverdi et al., 2018].

1.5 Overview of Thesis

This thesis is organized as follows. In Chapter 2, a review of previous work is presented. In Chapter 3, we provide an overview of our system, followed in Chapter 4 by an explanation of how our system generates an initial 3D model from GIS data. Chapter 5 presents a method to split large meshes into smaller pieces that can fit inside the 3D printer. In Chapter 6, we describe the various methods our system uses to create a more recognizable and more

realistic landscape model. Chapter 7 presents our results and a discussion of their accuracy. We discuss our implementation details in Chapter 8. Finally, Chapter 9 summarizes the thesis and discusses possible future works.

Chapter 2

Background and Related Works

In this Chapter, we make note of the works most related to our system. Since our work combines geospatial data sets, 3D modeling and 3D printing into an integrated system, we present works related to these subjects along with an overview of 3D printing technologies and their limitations.

2.1 Geospatial Data Sets

Geospatial data sets form a huge body of information that can be used to study the world that we live in. To embed such vast amounts of data into a single integrated location, the Digital Earth framework was proposed [Goodchild, 2000]. This framework enables the integration of different types of geospatial data captured from different devices, such as satellites, UAVs (Unmanned Aerial Vehicles) or mobile devices with a GPS sensor [VTP, 2015, Mahdavi-Amiri et al., 2015a]. Data types available in Digital Earth frameworks are generally categorized into four groups (Figure 2.1). Each group is described in the following:

- **Raster Data:** This data type stores information in the cells of a grid. Raster grids are appropriate for storing continuous data such as temperature. Satellites and UAVs with DSLR cameras are examples of devices that can capture data in this format. Digital Elevation Models (DEM), satellite images and orthophotos are the most commonly used examples of this data type.

- **Vector Data:** This data type is formed of vertices and paths. The vertices are coordinates and paths define the connections between these vertices. There are three basic types of vector data: points, pathways and polygons (areas).

Generally, vector data of any of the aforementioned types can be tagged with contextual information. Examples include specifications of roads, rivers or political boundaries. GPS sensing devices are one of the sources that can be used to capture vector data, such as by carrying a GPS device while moving along a hiking trail. This is the primary source of data collection for the free collaborative service known as OpenStreetMap, which provides vector data for the whole world.

- **Point Clouds:** Laser detection systems (such as LiDAR) use laser light to densely sample a region of the globe. The sampled data from these systems create large sets of 3D points referred to as point clouds. These point clouds are only available for limited regions of the world and capturing them requires a huge amount of resources. The raw point clouds are not generally usable directly, and a set of post-processing algorithms should be applied to them to create meaningful data, such as 3D models, out of them.

- **3D Content:** 3D models of buildings, vegetation and bodies of water are examples of 3D content. There is currently no device that can directly capture them, so they are either created manually, interactively [Ketabchi et al., 2015, Longay et al., 2012] or by using post-processing algorithms on LiDAR point clouds [Wang et al., 2008, Sun and Salvaggio, 2013].

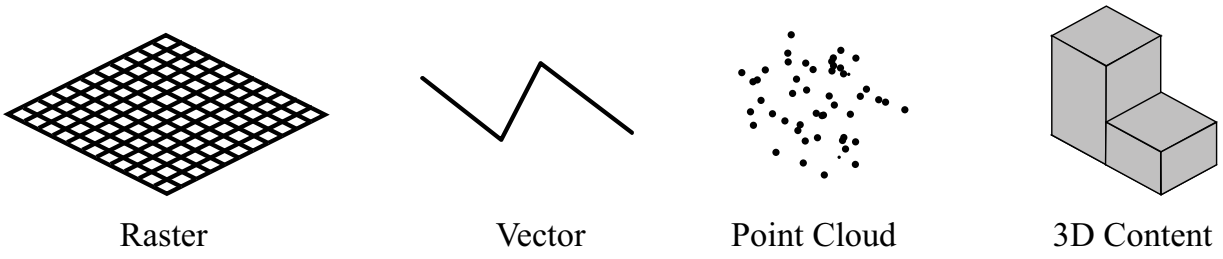


Figure 2.1: Data types available in Digital Earth frameworks.

Such geospatial data sets are captured through different techniques (e.g. satellites, LiDAR, surveying, or drones) [Campbell and Wynne, 2011]. Much effort have been devoted to clean-up, post process, and visualize these data sets [Mahdavi-Amiri et al., 2015a, Musialski et al., 2013, Sellers et al., 2013, Cozzi and Ring, 2011, Cozzi and Bagnell, 2013, Djavaherpour et al., 2017]. Although a vast variety of available geospatial data exists that can be used to represent a location, much work has gone into synthesizing geospatial data in order to obtain a better visualization, correct the available data sets, or build a virtual environment. For instance, a comprehensive survey on how to construct 3D buildings using point clouds is presented in [Musialski et al., 2013]. Road systems can be also generated using sketch-based systems [Applegate et al., 2012, McCrae, 2008]. Terrains can be synthesized using Digital Elevation Models (DEM) [Zhou et al., 2007] or multiresolution approaches [Brosz et al., 2008]. A variety of different data sets are also integrated in [Samavati and Runions, 2016, Ketabchi et al., 2015] in an interactive system that corrects the mismatches between the elevation data sets and imagery. In [Samavati and Runions, 2016, Ketabchi et al., 2015], in addition to employing geospatial data to determine the location of the geospatial features, some features are also synthesized to produce better representations for roads, trees, and water bodies.

2.2 3D Modeling

3D modeling is a general term describing methods to create virtual representation of objects. Here we discuss 3D modeling in the context of Digital Earth and creating and segmenting 3D content.

To create a 3D model of a region on the Earth, [Ketabchi et al., 2015] integrates DEM, satellite images and input sketches. In [Manferdini and Remondino, 2010], highly-detailed 3D models found in the Cultural Heritage field are segmented into subcomponents using both automatic and manual methods to facilitate annotation and data retrieval in a web-

based visualization. Some 3D modeling methods focus on creating 3D content using images and footage of a real world environment. These methods generally rely on computer vision and photogrammetry techniques to reconstruct 3D models of the input data. In most cases, using these methods requires large data collections or appropriate photos with detectable features [Schöning and Heidemann, 2015]. The more recent methods in this category focus on creating 3D models out of point clouds retrieved using LiDAR technology [Sun and Salvaggio, 2013, Wang et al., 2008].

Based on the type of input data, there are different methods one may use to create 3D content. To create a 3D model out of input sketches in [Olsen et al., 2011], an initial mesh is created by using Constrained Delaunay Triangulation (CDT), followed by a subdivision to better fit the input sketch. In [Ketabchi et al., 2015], an initial 3D model of a region on the Earth is created using a DEM, and using input sketches the system can be guided to apply corrective deformations to problematic regions caused by the inaccuracy of the DEM.

Another method to create 3D content is to use advancing front mesh generation techniques. Using these methods, a mesh is created out of a boundary by cutting off elements one by one. In [Schöberl, 1997], by describing a set of abstract rules for cutting off elements, a method for surface and volume mesh generation was provided.

In other methods, by algebraically combining parametrized boundaries, an interior grid mesh can be created [Halbert et al., 2017, Thompson et al., 1985]. For volumetric input data, marching cubes is a method to create a 3D model that traverses voxels in 3D space and creates appropriate triangles for each of those voxels [Lorensen and Cline, 1987]. In [Hilton and Illingworth, 1997], unlike marching cubes, the 3D space is not decomposed volumetrically and a surface mesh is created by placing mesh vertices according to local surface geometry. In this work, a 3D Delaunay constraint is also used to ensure the creation of a high quality mesh.

Methods that work on point clouds (e.g. from LiDAR), generally try to segment the

input point clouds into meaningful regions and then create appropriate 3D content for each region separately. In [Wang et al., 2008], this segmentation is done using the active contour algorithm to separate trees from the underlying terrain model. In this work, LiDAR point clouds are used to create individual tree models. To create these models, the point cloud is projected into several 2D slices and then, using morphological operations, the boundary of each tree is found for each slice. Layers are then created by giving a thickness to the tree boundary in each slice. The final 3D model of a tree is later created by stacking all of these layers. In [Sun and Salvaggio, 2013], to create 3D building models, a segmentation is performed using a graph cuts optimization followed by a clustering to group building points together. The final 3D model for each building is then created by using a 2.5D dual-contouring approach.

2.3 3D Printing

In this section, we first review a number of 3D printing technologies. We then list a number of works that focus on using 3D printing.

2.3.1 Technologies

3D printing is considered a rapid prototyping method that enables fast creation of physical models. There are many different technologies that are used for 3D printing. In the following, we discuss a number of these technologies.

Stereolithography (SLA)

SLA is a liquid-based method where a photosensitive polymer is solidified using an ultraviolet laser to manufacture a 3D model layer by layer. This solidification is done using a process called photopolymerization in which chains of molecules are linked together using light. This technology requires a rather high amount of maintenance as several parts of the printing machine need to be replaced regularly, e.g. its material tank and build platform.

Inkjet 3D Printing

Inkjet 3D printing, also known as binder jetting or 3DP, is a process where a liquid binder is used to activate a powder such as starch and gypsum plaster. Based on the material used, full color printing is possible in this technology. Based on the physical model geometry, parts created using this approach may become brittle and require careful handling. Currently, there is no desktop version of 3D printers that use this technology, which makes it less accessible and quite expensive to use.

Fused Deposition Modeling (FDM)

In FDM, a thin filament of plastic is melted and extruded through a nozzle to create layers of a physical model. Due to the development of cheap FDM printers and cheap materials this technology is widely accessible. FDM machines do not require a lot of maintenance and are easy to work with. There is a wide range of materials with different colors and textures that are available for this technology. A number of these materials are listed in Table 2.1. Similar to other printing technologies, FDM suffers from a number of problems. In Table 2.2, we list a number of these problems most related to our system.

2.3.2 Related Works

Many applications have been proposed for 3D prints. For instance, they have been used to recreate cultural heritage sites [Scopigno et al., 2014], to make 3D puzzles [Xin et al., 2011, Lo et al., 2009], or produce objects that can stand or spin [Prévost et al., 2013, Bächer et al., 2014]. In [Djavaherpour et al., 2017], a scale model of the world was created to help visualize geospatial data sets that span large regions of the globe. Our system considers a new application for 3D prints (i.e. landscape creation), which, unlike [Djavaherpour et al., 2017], can show detailed information for much smaller regions, like cities or mountains.

Although 3D prints have many applications in fabrication, they still have some limitations. For instance, much research has been done to strengthen 3D prints by fortifying

high-stress parts of the print [Telea and Jalba, 2011, Stava et al., 2012, Zhou et al., 2013]. Since the result of the 3D printing is usually stiff and static, some related work is devoted to providing elasticity and movement to the final 3D printed object [Panetta et al., 2015, Schumacher et al., 2015, Pérez et al., 2015].

Since we wish to use affordable and small 3D printers, we must grapple with two significant challenges from 3D printing: limitations on the number of colors and materials as well as limitations on the size of the printing volume. In the following sections, we review works in several research areas that address these limitations along with other considerations for 3D printing. One area is focused on designing models that are appropriate for 3D printing while overcoming material limitations. Reducing material waste is another research area that aims to reduce use of support material to hold the printed model. The last area focuses on segmenting 3D models for printing based on different objectives, including the size limitation of 3D printers.

2.3.3 Design

In order to produce a better looking 3D print, in [Mahdavi-Amiri et al., 2015b] a system was proposed to segment a model into nearly developable patches that can be used as a guide for pasting desired materials on top of a 3D print. Some systems have also been designed to print a texture on an intermediate domain such as water [Zhang et al., 2015, Panozzo et al., 2015] or plastic [Schüller et al., 2016] and then paste it on the 3D print using a simulation of the intermediate domain. In [Savage et al., 2014], a technique is provided for routing pipes inside 3D models to create interactive objects, such as a radio or a haptic feedback rabbit. For material-aware fabrication, a data-driven approach is presented in [Yang et al., 2015] to reform shapes based on the fabrication material.

2.3.4 Material Waste

To avoid wasting material, new methods of packing a segmented 3D object with less unused space and therefore less need for supporting material have been proposed in [Vanek et al., 2014b, Yao et al., 2015]. In [Wang et al., 2013], a skin-frame structure is created to reduce the material usage for the interiors of 3D models. To reduce support material usage for FDM based 3D printers, an optimization framework is presented in [Vanek et al., 2014a], which progressively creates a support structure for overhang points, while attempting to minimize the overall length of this support structure.

2.3.5 Segmentation

To overcome the problem of the size of 3D printers, Chopper was proposed to segment a 3D print into printable pieces that can be later attached to each other using some male and female pins [Luo et al., 2012]. Other systems include CofiFab [Song et al., 2016], in which a laser-cut base is constructed by coarsening the mesh. The mesh is then segmented and attached to the base. As a result, the 3D printing time and material usage are reduced and larger objects can be printed due to the segmentation of the initial mesh. In this thesis, we are concerned with segmenting a large surface representing a landscape region and, unlike Chopper, we do not need to account for arbitrary meshes. As we have different geospatial features in a landscape region, we can segment each of their corresponding meshes separately, which results in a more appropriate segmentation. We use a grid segmentation for each of these features while respecting a cost function that minimizes both the total number of pieces and the number of small pieces. We also solve the limitations on color by creating separate extruded layers for different geospatial features.

Material	Description
PLA	PLA (Polyactic Acid) is a cheap thermoplastic (i.e. becomes liquid at certain temperatures) which is made from renewable materials such as corn starch, tapioca or sugarcane. PLA is considered a bio-degradable material as it is plant based.
ABS	ABS (Acrylonitrile Butadiene Styrene) is a thermoplastic which is relatively more expensive than PLA. Models created with ABS are strong and have a little flexibility.
Nylon	This material is a PA (Polyamide) thermoplastic which has good mechanical properties. As it melts at about 250 degrees Celsius, it requires a 3D printer that can heat its extruder to this temperature.
T-Glase	The chemical name of this material is Polyethylene Terephthalate (PETT). This material is able to produce a glass-like solid.
Wood Filament	This material is created by mixing wood particles with PLA along with a polymer that binds them. This material can create models that have a wood-like surface.
Flexible Filament	A flexible filament can create a rubber-like model. However, this type of material is difficult to work with and requires some replacement of 3D printer parts to support printing with this material.
Conductive Filament	By mixing graphene with PLA, conductive materials are created. This type of material can be used to 3D print simple electric circuits.

Table 2.1: Materials available for use with FDM technology.

Limitation	Description
Low Accuracy	Based on printing settings, the Z axis of printing can have a low accuracy and will create a stepping effect.
Overhang	Parts of the model hanging in the air (i.e. overhangs) require additional support structures to hold them while printing. Removing this support structure completely is a difficult task when using only one extruder to print both the model and the support structure. It is also troublesome to remove support material from very small parts of a model, as it may break those parts.
Warpage	As FDM works by heating a plastic, it can result in warping the extruded material. Using a heated bed or some tape can help reduce this warpage. However, the larger a part grows horizontally, the more this warpage increases [Armillotta et al., 2018].
Printing Area	FDM, similar to other RP based methods, have 3D printers with limited printable area.
Limited Material	FDM printers have a limited number of materials they can use during a single printing session.

Table 2.2: FDM technology limitations.

Chapter 3

System Overview

As discussed in Chapter 1, the 3D printing of scale models using geospatial data sets is a challenging task. In order to create this physical model, we developed a modeling system with several components. In this chapter, we briefly discuss the major parts of our system and how they help to create a physical model. We give an overview of these parts and how they work together to solve the challenges faced as part of creating, printing and assembling physical models of landscapes.

To print a physical model of a geospatial regions, the region of interest on the globe is taken as input (see Figure 3.1). Our system then retrieves the required geospatial elevation data and features corresponding to the region based on the user's needs from a preselected set of sources, including OpenStreetMap (OSM) and Canadian Digital Surface Model (CDSM). Some examples of these features include vegetation areas, roads, lakes and buildings. Elevation data is stored as a regular grid of data points, while features are in vector format, representing a path (e.g. roads) or boundaries (e.g. lakes).

3.1 Mesh Generation

Our system creates an initial 3D mesh that respects the elevation data and the features. This is done by creating a 2D triangulation that respects the features and then updating each point of the triangulation with its respective height using the elevation data. This 3D mesh is then processed into a set of submeshes for each feature. These submeshes are extruded as layers for 3D printing.

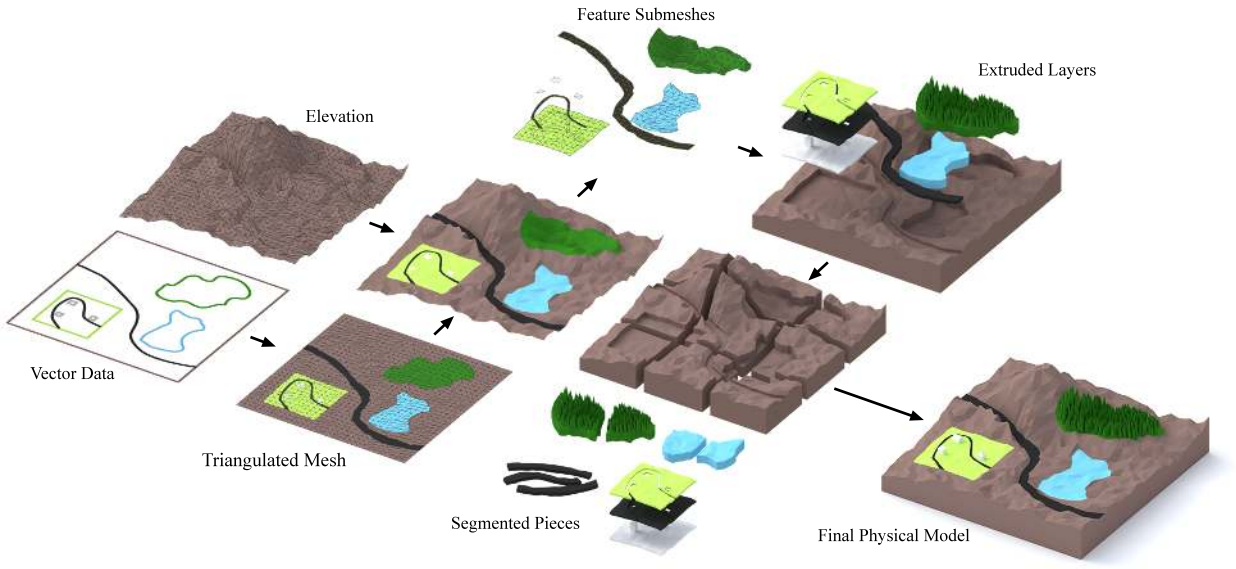


Figure 3.1: We use several GIS sources to retrieve the required features and elevation data for the region of interest. A Constrained Delaunay Triangulation is employed to create an initial 3D model. Using a technique based on winding numbers, feature submeshes are extracted from this model. These submeshes are used to create extruded layers. Each of these layers is designed based on the characteristics of its corresponding feature to create a more realistic model. For extruded layers that do not fit inside the 3D printer, a grid segmentation method is used to create appropriate smaller pieces. Eventually, printing all the pieces and assembling them creates the final physical model.

3.2 Segmentation

If a layer does not fit inside the 3D printer, our system applies a segmentation algorithm to create smaller printable pieces. A non-linear optimization is solved to minimize a cost function that will prevent creation of very small pieces. A post-processing step will attempt to further merge smaller pieces in order to create more desirable pieces.

3.3 Adding Details to Geospatial Features

In the final phase, our system creates a 3D model file after adding extra details to each piece to assist with feature recognition. To model vegetation features, our system runs a simulation and applies a displacement map to its mesh. For urban structures that have delicate features with complex interactions, our system provides a layering mechanism to overlay additional layers (i.e. stencil layers). Our system also provides different roof designs for modeling buildings to create more realistic feature layers.

3.4 Assembly

Assembling all of the 3D printed models creates the final physical model of the desired region. Our system provides offsetting to account for 3D printer inaccuracies and make assembly of the printed pieces easy. Each step of this process, as illustrated in Figure 3.1, is described in the following chapters.

3.5 Summary

In this chapter, we gave an overview of the various components of our system. We discussed how our system generates 3D models from GIS sources and how it handles the size limitation of 3D printers using segmentation. Furthermore, we described how our system adds details to geospatial features to better represent them along with how our system helps to speed up assembling 3D printed parts.

Chapter 4

Mesh Generation

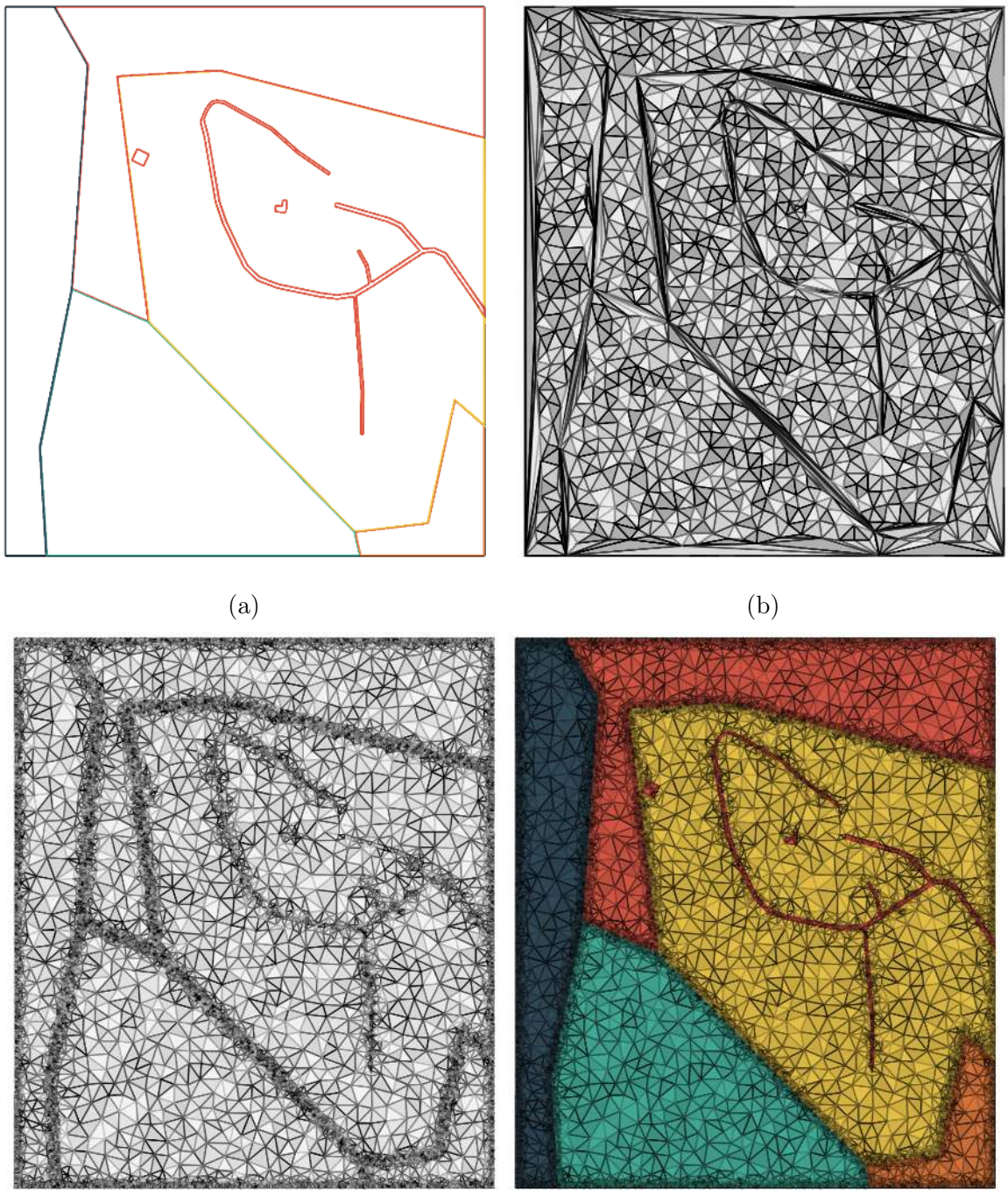
In this chapter, we discuss the process our system employs for generating the 3D model for a region of interest using GIS sources. We also discuss how our system creates separate 3D models for each feature in order to print them with a different material.

3D printing physical models starts with the creation of 3D virtual models. Using a 3D printing software, 3D models are turned into specific commands that a 3D printer can interpret to fabricate a physical object. Since we do not have 3D models of landscapes for the whole globe, our system first needs to generate a corresponding 3D model for the desired region that needs to be printed.

4.1 Triangulation

In order to 3D print an object, a 3D model has to be created based on the GIS data. Although creating a regular 3D mesh from the elevation data is straightforward, constraining the model to respect feature data requires more consideration. To create this 3D model, we first generate a 2D mesh by employing Constrained Delaunay Triangulation (CDT) [Chew, 1987], with the geospatial vector features as constraints (see Figure 3.1). We chose CDT to preserve the resolution of the geospatial vector features without increasing the resolution of the mesh globally.

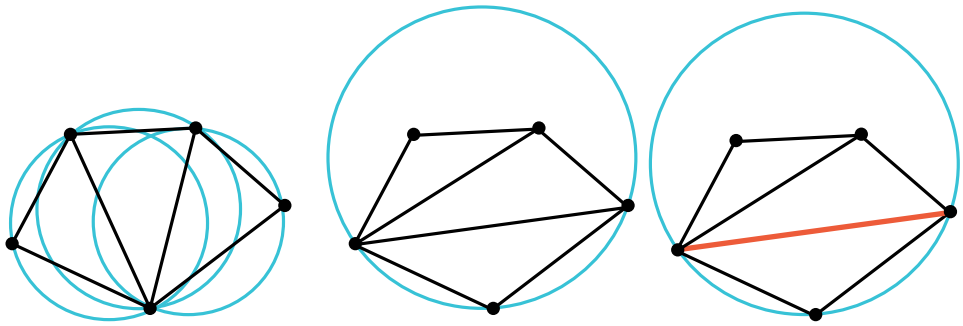
A triangulation of point set P is a Delaunay triangulation if no points of P are inside the circumcircle of any triangle (see Figure 4.2). An extension of Delaunay triangulation is Constrained Delaunay triangulation (CDT), where in addition to points, the input may contain a set of line segments called constraints, which they become the edges in the triangulation. As these constraints fix several edges of the triangulation, it may not be possible to achieve



(a) (b)

(c) (d)

Figure 4.1: For the input features in (a), we can create the CDT in (b). In (c), by restricting edge lengths, we ensure the resolution of our mesh is more than the resolution of the input elevation data. As described in Section 4.2.2, the triangles in this mesh are marked according to their corresponding features (d).

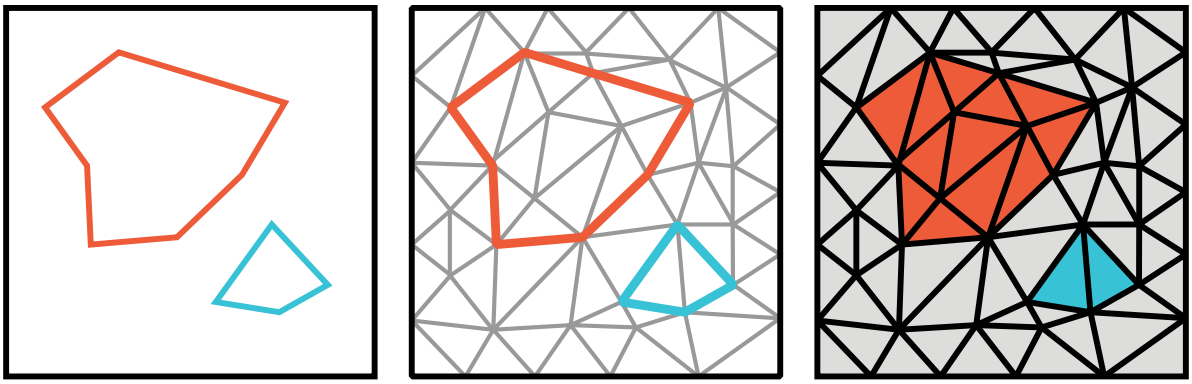


(a) (b) (c)

Figure 4.2: A triangulation has the Delaunay property if all the circumcircles of the triangles are empty (a). If there exists a circumcircle that contains other points, the triangulation is no longer Delaunay (b). In CDT, the points that are not visible from a triangle (i.e. they are across a constrained edge) are ignored. The triangulation in (c) is a CDT where the constraint is shown in red. Due to this constraint, the circumcircle does not contain any visible points, and therefore, the triangle satisfies the Delaunay property.

the Delaunay property in CDT. Therefore, in CDT, the circumcircle of triangles only checks the points that do not cross a constraint (i.e. they are visible from the points of the triangle) for the Delaunay property (see Figure 4.2c).

There are several approaches to implementing CDT. We have used an incremental approach [De Floriani and Puppo, 1992] to create our desired mesh by adding constraints incrementally. In this approach, every time a point or constraint is inserted in a CDT, the triangulation is updated accordingly. In this process, we also restrict edge lengths to an appropriate limit to make sure the resolution of our mesh is not less than the resolution of our elevation data (see Figure 4.1c). Adding elevation data to this triangulation produces our initial 3D model as a mesh (Figure 4.3).



(a) (b) (c)

Figure 4.3: For any number of features (a), we use CDT to create an initial mesh (b). A region growing algorithm is applied later to determine what feature each face belongs to (c).

4.2 Creating Separate Feature Layers

Our initial 3D elevation mesh can be used to identify the peaks and valleys of the selected region. However, geospatial features are not necessarily visible in this model. Those features describe real-world entities, such as roads, buildings or vegetation areas, and are important for understanding a region of interest. To help in their identification, we can use a separate and appropriate material with a natural and familiar color (e.g. translucent blue for lakes

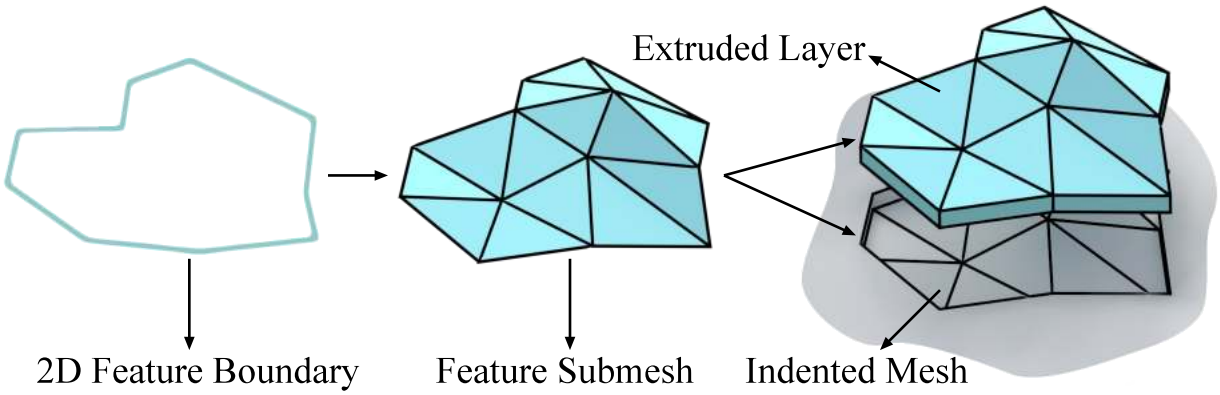


Figure 4.4: To allow the display of more information, we create separate layers for each feature.

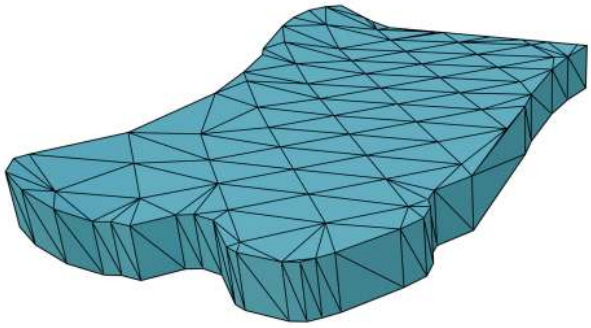


Figure 4.5: An example of extruding feature submeshes vertically. The extrusion is done by adding two vertical triangles for each boundary edge of a submesh.

or bodies of water) to print the feature region. Alternatively, rather than using a separate material, we can extrude those regions to make them visible on the mesh. However, this solution requires a guide to describe each feature and may result in confusion when identifying features.

In order to use a different material for each feature, we create a set of *feature submeshes* and then extrude each of these submeshes to create a set of *extruded layers* (Figure 4.4). We extrude the submeshes vertically to avoid overhangs on the edges of printed layers. This will increase the accuracy of layer edges and helps with fitting them in the final physical model (see Figure 4.5). A detailed discussion is provided in Chapter 7. These extruded layers can be later attached by indenting the underlying mesh. We can also add details to each of these extruded layers based on the properties of the feature associated with that feature layer, as described in Chapter 6. Eventually, these extruded layers should fit together nicely into the indented mesh to represent the final model.

4.2.1 Extracting Feature Boundaries

Geospatial features are represented as a set of points, pathways or areas tagged with contextual information to identify their corresponding entities. To create extruded layers for each feature, our system begins by extracting submeshes for all of our input geospatial features.

For each of these features, our system obtains a vector of 2D points defining the boundary of an area that a feature occupies in 2D. In the case of pathway features, as described in Section 6.1, we transform them into area features by applying a slight offsetting. This is followed by extracting boundary loops as closed paths from the vector of points, and determining whether each boundary loop is an outer boundary or an inner boundary. We assign a counter-clockwise direction for outer boundaries and a clockwise direction for inner boundaries. These directions help us define the inside region of each feature. We input all of the directed boundary paths for each feature to CDT, and retain this direction information for each edge that corresponds to a feature boundary in the resulting triangulation.

4.2.2 Feature Interior Marking

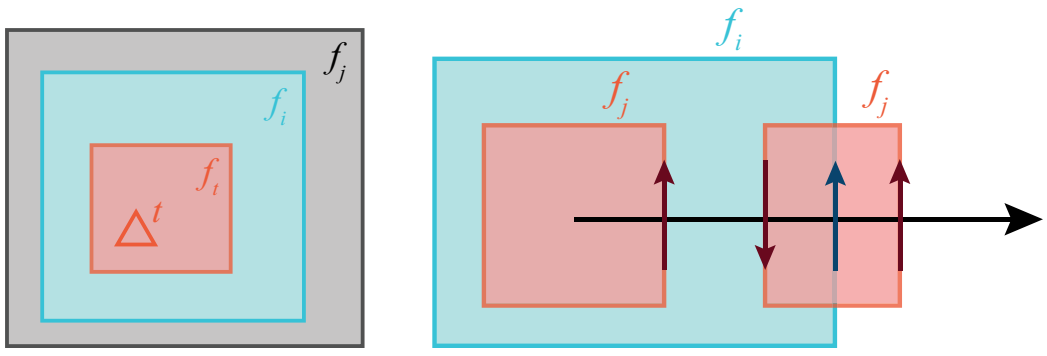
To create feature submeshes, it is necessary to know the interior triangles encompassed by the feature vectors. These feature vectors are a collection of line segments (Figure 4.3b), and their interior triangles are not readily available; that is, extracting features submeshes out of these vectors is not a trivial task. Feature submeshes may overlap or contain other feature submeshes (e.g. buildings inside a vegetation area). In the case of feature submeshes contained in other feature submeshes, we need to extract the *containment hierarchy* of the feature submeshes. This hierarchy is a tree with the root node being the base feature that contains everything (e.g. ground). In this tree, a feature is a child of another feature only if it is contained in that feature. This containment hierarchy is beneficial in several ways. For example, we can exclude roads that are outside a residential region from being printed, or we can use it to create a stencil layer, as described in Section 6.3, to print all the feature submeshes contained in a residential area. Hence, a robust method is needed to identify the feature hierarchies for the feature submeshes. To achieve this, we perform a region growing algorithm that extracts the containment hierarchy and accounts for overlapping feature submeshes.

We denote the list of all the triangles in the mesh as M, and the i_{th} feature as f_i. The

feature submesh M_i represents the submesh of M which is inside f_i. We say f_i is contained in f_j if and only if all of the triangles in M_i are also present in M_j. Based on this definition, we create a tree T that represents the hierarchy of feature submeshes, where each node of this tree is a feature boundary, and the feature boundary f_i is a child of f_j if and only if f_i is contained in f_j. For a given triangle t, we let f_t denote the smallest feature that contains t (Figure 4.6a). We can then find the hierarchy of feature containment by traversing all the ancestors of the node corresponding to f_t in T. As an example, the containment hierarchy for the features shown in Figure 3.1 is illustrated in Figure 4.7.

To determine feature containment and also the smallest feature for every triangle in the mesh, we use a technique based on winding numbers [Hormann and Agathos, 2001]. The winding number is one of the methods used to check the inclusion of a point in a polygon. It shows the total number of counterclockwise turns of a curve around a given point. More formally, let us use polar coordinates for a closed curve and translate our coordinate system to set our query point as its origin. Then the curve is defined as:

$$r = r(t) \quad \text{and} \quad \theta = \theta(t) \qquad \text{for} \quad 0 \leq t \leq 1.$$



(a) (b)

Figure 4.6: In (a), the red feature submesh is inside two other feature submeshes. To detect this situation, as shown in (b), we cast an infinite ray to determine the hierarchy of surrounding feature submeshes. Each arrow here shows a crossing for a feature boundary.

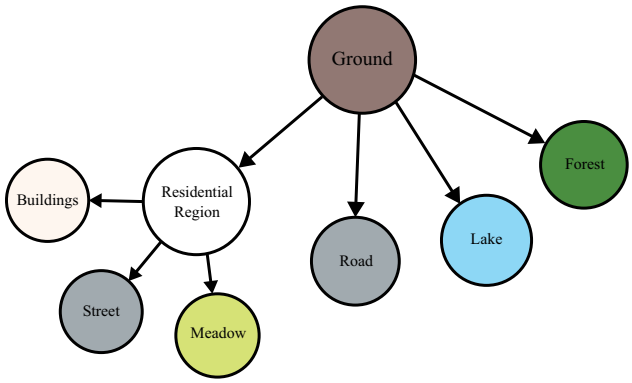


Figure 4.7: An example of hierarchy containment for the features shown in Figure 3.1.

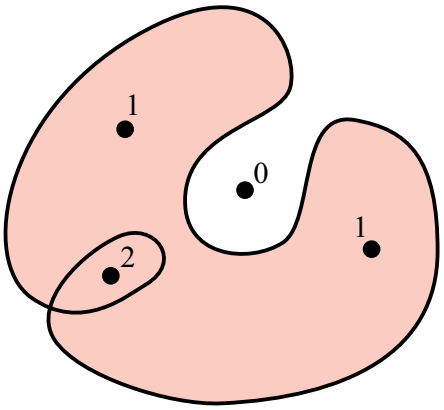


Figure 4.8: An example curve and winding numbers for several points.

Now, the winding number is defined by:

$$\frac{\theta(1) - \theta(0)}{2\pi}$$

This number is an integer, as we only consider closed curves (i.e. the starting and ending points are identical). Using this number we can decide if a point is inside a curve by checking if this number is greater than zero (see Figure 4.8). We expand this definition of winding numbers in our algorithm to extract the feature containment hierarchy.

Our algorithm begins by traversing all triangles in M. For each triangle t, we cast an infinite ray from the center of the triangle. If we cross an edge on the boundary of feature f_t, similar to the winding number method, we determine if the ray is exiting the feature based on the edge direction. In this case, we check if we have a node for f_t in our tree T. If so, we

return f_t as the smallest feature that the triangle t is contained in. If there is no node for f_t, we continue traversing the infinite ray to find all the edges intersected by the ray that belong to a feature boundary (see Figure 4.6b). For each of these edges, we assign a winding number w_{f_i}, based on the direction of the corresponding vector in the feature boundary f_i. We create a sequence of these winding numbers, starting from the closest crossing. An example sequence for the feature boundaries in Figure 4.6b can be seen below:

$$(+1_{f_i}, -1_{f_i}, +1_{f_j}, +1_{f_i}).$$

We reduce this sequence by cancelling each -1 with the *next* $+1$ for the same feature boundary. For instance, the example sequence above reduces to:

$$(+1_{f_i}, +1_{f_j}).$$

This sequence shows the hierarchy of feature submeshes surrounding this triangle (f_i, f_j). We apply this process for all triangles to both create the hierarchy tree T and find the smallest feature submeshes that contain each triangle. As explained in Section 4.2, these feature submeshes are used to create extruded layers (Figure 4.3c).

Handling Partial Covering: Geospatial features can naturally overlap (see Figure 4.6b). In our physical models, only the surface of a region is visible and we need to choose one feature, out of all the overlapping features, for each triangle to belong to. This is not a trivial task, as there is no information in our input data about the feature type precedence. To address this issue, we offer two possibilities. First, we use a priority list defined by the user, and in case of overlaps these priorities are used to decide which feature submesh should be chosen. An example is when a river feature overlaps a road feature. In such situations, we can decide if the road is going over the river, as a bridge, or goes under the river, in the case of an underground tunnel. In each case, the surface material would be assigned to either the river feature or the road feature. If the priority is not explicitly defined, we utilize a statistical procedure to determine which feature has top priority by checking the corresponding

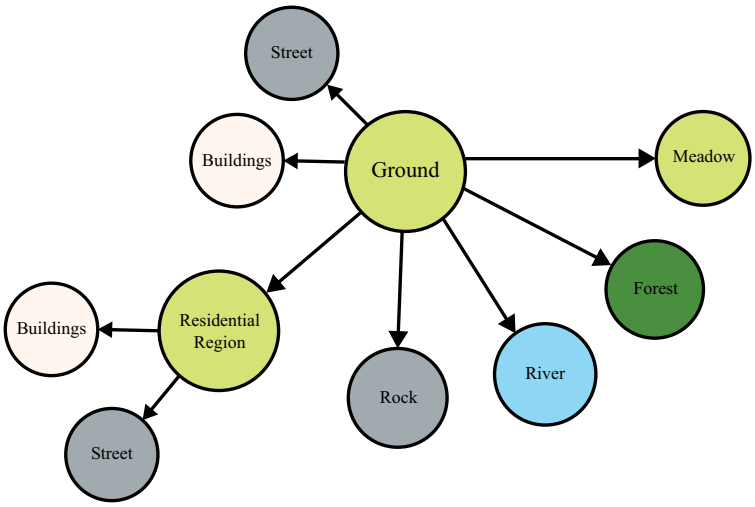


Figure 4.9: The containment hierarchy for our Lauterbrunnen result.

feature boundaries, summing the edge lengths for those edges along the boundary of the overlap, and choosing the one that has the longest total length. From this, we choose the dominant feature submesh and ignore small overlapping feature submeshes.

Figure 4.9 illustrates the containment hierarchy created by our system for our Lauterbrunnen physical model discussed in Chapter 7.

4.3 Summary

In this chapter, we first discussed how our system generates an initial mesh out of input GIS data. Using CDT, we combined both elevation and vector geospatial features data into a single mesh. We later described a region growing algorithm to find the hierarchies of different feature regions. Using this info, we can split the initial mesh into separate feature submeshes. Each feature submesh can be printed using a different material based on the characteristics of its corresponding feature submesh. In the next chapter, we discuss what happens when a feature submesh is larger than the printable volume of the 3D printer. We present a segmentation algorithm that splits these larger meshes into smaller printable pieces.

Chapter 5

Segmentation

After generating each feature submesh, we need to create an extruded layer that can be printed with a different material. However, if a feature submesh is larger than the printable volume of our 3D printer, we need to break it into smaller pieces first, and then we can extrude them to create extruded pieces for printing. We propose a method to create these pieces such that they fill the printable volume as much as possible (see Figure 3.1). Although our method is extendable to volumes, in our application, we mainly encounter feature submeshes that only exceed the 2D plane limitation. Furthermore, in those rare cases that a feature submesh extends beyond the height limitation, segmentation is trivial and does not require a sophisticated method. Hence, in our method we ignore the height limitation and project feature submeshes into the 2D plane for segmentation. In this chapter, we explain our segmentation method in detail.

5.1 Local Grid Segmentation

The largest printable area takes different shapes and sizes in different 3D printers. These shapes primarily include squares, rectangles and circles. To simplify the segmentation, for all of these cases we consider the largest square that fits within the printable area of the 3D printer to be the maximum size of a printable mesh. We later describe how to employ the unused areas to optimize the printing process in Section 5.2.

It is natural to reduce the number of segments, similar to the polygon interior covering (PIC) problem, where the aim is to cover a target polygon with the smallest units of a given convex polygon. This problem is known to be NP-hard [Culberson and Reckhow, 1994]. In PIC, the only criterion is the number of convex polygons used for covering. In our case,

we use squares to cover each feature submesh. Unlike PIC, after covering the whole feature submesh, we need to evaluate every resulting piece to see how appropriate they are for 3D printing (Figure 5.1).

To cover each feature submesh, we create a regular grid with each cell having the largest square size that is printable. By placing each mesh over this grid, we can break it into smaller pieces that fit inside the 3D printer. Since we use a local grid for each feature submesh, we can independently translate and rotate the mesh to find different ways to slice it. We call each translation/rotation pair a *configuration* in our algorithm. Based on the specific requirements of 3D printing, we define a cost function to evaluate each configuration and choose the optimal one (Figure 5.2). In the following section, we discuss how we perform this task and discuss the cost function.

5.1.1 Cost Function

There are many different metrics to consider when segmenting a feature submesh into a set of smaller pieces. We are interested in creating pieces that are not too small, as they are harder to fit into the final printed model and are also more fragile. To prevent creating small pieces, we define the cost function as,

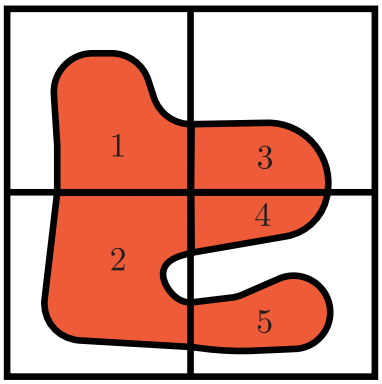


Figure 5.1: Each piece is a connected component of a feature submesh after segmentation. We need to evaluate each piece based on how appropriate they are for 3D printing.

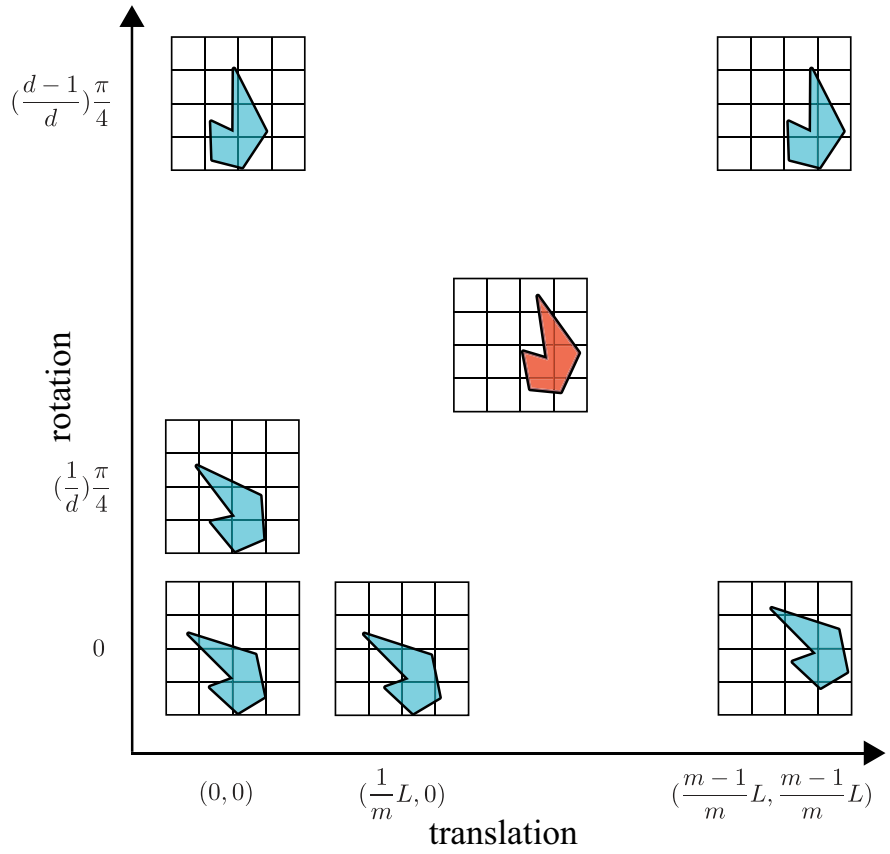


Figure 5.2: For each orientation of a feature submesh on a regular grid, where L is the grid cell size, we search different possible translations, shown as a 2D vector in the figure. To speed up this search, each translation axis is uniformly discretized to m values and the space of orientations is uniformly discretized into d values. We choose the optimal configuration for breaking the submesh into smaller pieces, shown as red.

$$C(r,t) = \sum_i \frac{1}{a_i},$$

where $C(r,t)$ is the cost of using the configuration with rotation r, and translation t and a_i is the area of each piece (Figure 5.1). Note that a_i is determined after uniformly scaling the feature submesh and the grid to make the sides of each grid cell unit length. We chose the inverse function to compute cost among several alternatives, as it is a simple and computationally appropriate option.

The upper bound of our cost function is infinity, as there is no limit to how small the pieces may become. To find the lower bound, we need to solve the following minimization,

$$\min_{(r,t)} \sum_i^N \frac{1}{a_i},$$

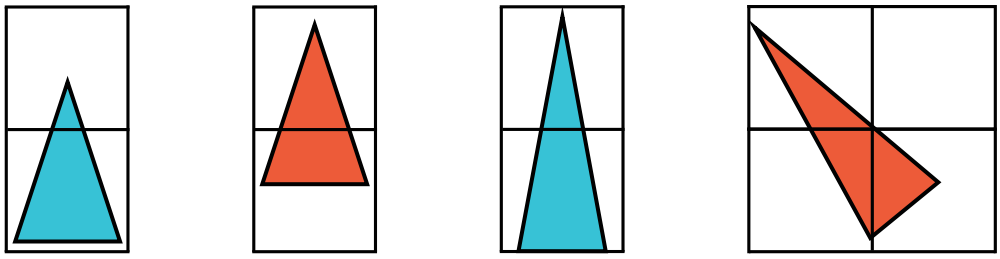
where we assume we have $N \in \mathbb{N}$ pieces. Solving this minimization leads to the answer $a_i = \frac{A}{N}$, if we denote the area of the mesh as A. Therefore, for a fixed number of pieces, we have,

$$\min \sum_i^N \frac{1}{a_i} = N \times \frac{N}{A} = \frac{N^2}{A}.$$

The cost function has hyperbolic growth as we decrease the areas of the pieces, and prevents creating small pieces. Moreover, if we increase N, the minimum cost will increase quadratically, which illustrates that the cost function also minimizes the number of pieces. As we know the minimum value for N (with unit length grid cell dimensions) is $\lceil A \rceil$, we can determine the global minimum:

$$\min \sum_i \frac{1}{a_i} = \frac{\lceil A \rceil^2}{A}.$$

This lower bound cannot be achieved for all shapes of a mesh. For example, in Figure 5.3a, the shape of the mesh allows one to create equal area pieces (Figure 5.3b). In Figure 5.3c, we see an example where creating equal area pieces with the minimum number of pieces is



(a) (b) (c) (d)

Figure 5.3: The shape in (a), can be repositioned on the grid to create equal area pieces (b). In (c), there is no way to create 2 equal area pieces with two grid cells, and achieving the lower bound cost is not possible. If we allow for the creation of more pieces, we can create 3 equal area pieces (d).

not possible. In some cases, it may be possible to create equal area pieces if we first break the mesh into more pieces (Figure 5.3d). However, in general creating more pieces is not desirable, and we should consider adding more pieces only if it results in a smaller cost for the whole mesh.

In order to see under which conditions the cost function will increase the number of pieces, we can perform an analysis of the cost function. We start by checking when it is possible to ensure that increasing the current number of pieces would not result in a smaller cost value. If, for the current configuration, $a_i \geq \frac{A}{r}$, where $r \leq 1$, and $M \geq N$ is the new number of cells, we have,

$$\sum_{i}^{N} \frac{1}{a_i} \leq \frac{Nr}{A} \leq \frac{M^2}{A} \Rightarrow$$
$$\frac{1}{r} \geq \frac{N}{M^2}.$$

Therefore, we can be sure that no better answer will be found by increasing the number of cells to M if $a_i \geq \frac{N}{M^2}A$. This suggests that, if there is a case in which increasing the number of pieces can prevent creating a very small piece, the cost function would allow for an increase in the number of pieces. As we discuss in Section 5.1.2, we attempt to search the

discretized space of configurations as much as possible, and if a better configuration with more pieces is found, we would consider that as well.

5.1.2 Finding the Best Configuration

To find the best configuration, we can rotate and translate each feature submesh on a regular grid with cells whose areas are each the same as the printable area of the 3D printer. The range of possible translations, as we are searching on a regular square grid, is:

$$0 \leq t_x, t_y < 1, \tag{5.1}$$

where t_x and t_y are translations along the x and y axes respectively. For the rotation, thanks to the symmetry of the square grid cells, we need to only rotate up to 45 degrees:

$$0 \leq r \leq \frac{\pi}{4}. \tag{5.2}$$

All possible combinations of r and $t = (t_x, t_y)$ define all the possible configurations.

Therefore, our problem is finding $\min C(r, t)$ over the aforementioned domain for r and t. This problem is a non-linear optimization. One way of solving this optimization is using iterative methods, such as gradient descent. However, this function has several undefined points where the area of the pieces reaches zero, which makes finding the global minimum of the function using iterative approaches troublesome. To avoid such problems, we solve this non-linear optimization by discretizing the ranges of both r and t to ten uniform steps each and performing a brute-force search. Since the segmentation time of feature submeshes is quite negligible compared to the 3D printing time, we decide on using this exhaustive search to avoid any probable local minima. Apart from creating more undesirable pieces, a local minima can potentially result in more 3D printing sessions. This leads to an increased overall 3D printing time due to the required preparation time for each session. Furthermore, the discretization resolution can be changed in our system if a higher accuracy is required. An example output of our segmentation can be seen in Figure 5.4.

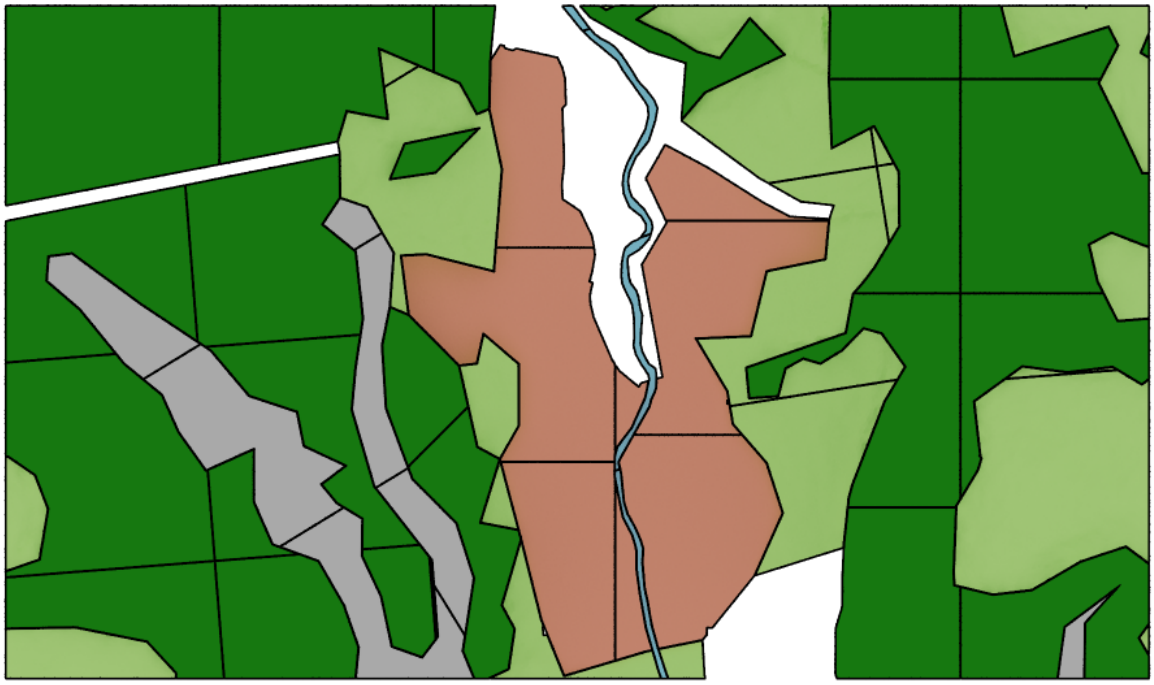


Figure 5.4: The result of our segmentation method for the features of our Lauterbrunnen result, discussed in Chapter 7.

5.2 Post Processing

A local grid can help to segment a feature submesh separately from the rest of the feature submeshes. However, as we limit the segmentation to use a single square grid for the whole feature submesh, creating a number of very small pieces might be unavoidable, as shown in Figure 5.5. One way of fixing this issue is to merge some of the resulting pieces locally after segmentation (Figure 5.6).

We start by creating a connectivity graph from the results of the global grid segmentation. Each node in this graph represents a piece, and we connect two nodes if the corresponding pieces are adjacent (Figure 5.7a). In order of highest cost, we traverse the nodes of the graph and merge all adjacent pieces until the resulting piece completely fills the full printable area (not the largest fitting square), and then repeat the process for all the remaining nodes in the graph (see Figure 5.7b). In this greedy approach, as we start with the smallest piece, we

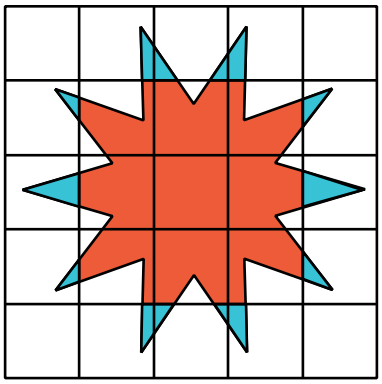
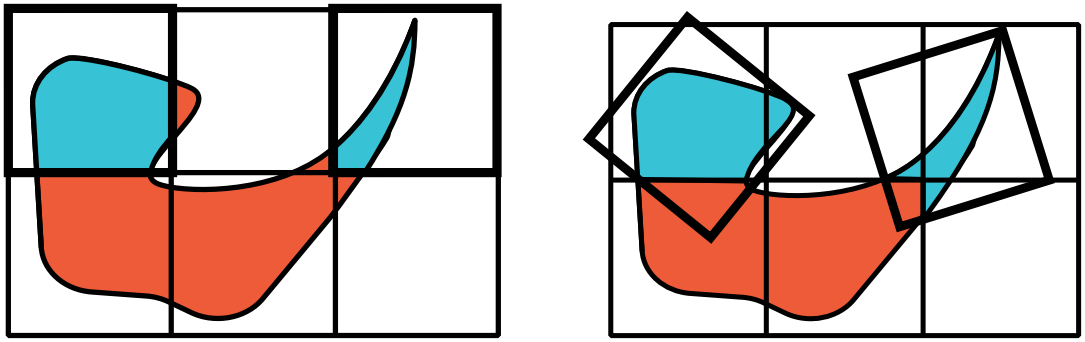


Figure 5.5: An example shape that, when using a global grid, unavoidably creates small pieces.



(a) (b)

Figure 5.6: The blue pieces in (a) after post-processing can include neighbor pieces as well, as in (b).

typically merge the most undesirable pieces first. Based on our cost function, merging these very small pieces first will have the most reduction in the overall cost of the segmentation.

5.3 Summary

In this chapter, we explained a method to split large feature submeshes to smaller printable pieces. We first used a local grid to split an input mesh into pieces that fit inside the 3D printer area. We proposed a cost function for searching different translations and rotations of the input mesh over this grid. We then analyzed how this cost function prevents creating

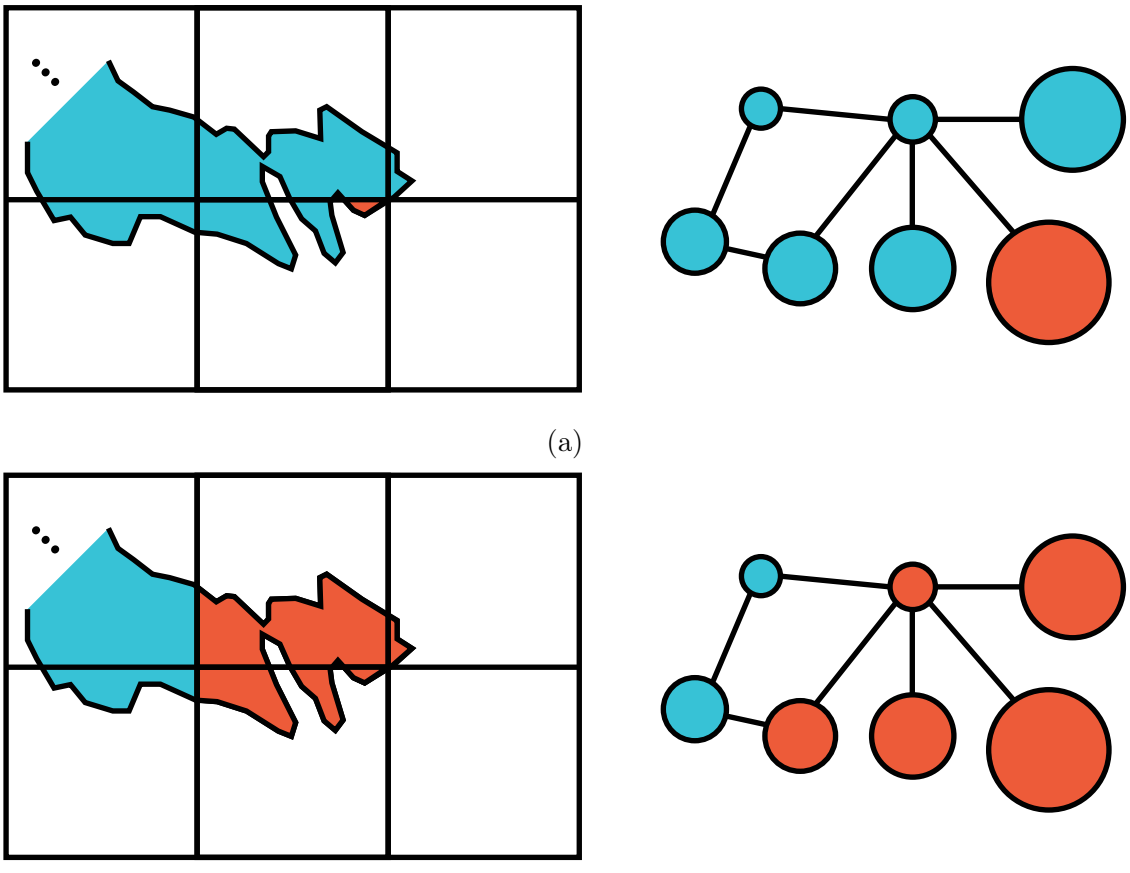


Figure 5.7: In (a), a part of a feature submesh and its corresponding connectivity graph is shown. The size of nodes in the graph is proportional to the cost of their corresponding pieces. The node with the highest cost is shown in red. In (b), we have applied our post-processing to merge adjacent pieces. The merged pieces are shown in red. Note that in this example, we assume the largest printable area of the printer is a rectangle slightly bigger than the grid cells, which makes this merging possible.

small pieces while also preferring to reduce the number of segmented pieces. In the last section, we described a post-processing method to merge some of the unavoidable small pieces locally after segmentation. In the next chapter, we discuss the different ways our systems uses to make more recognizable and more realistic pieces.

Chapter 6

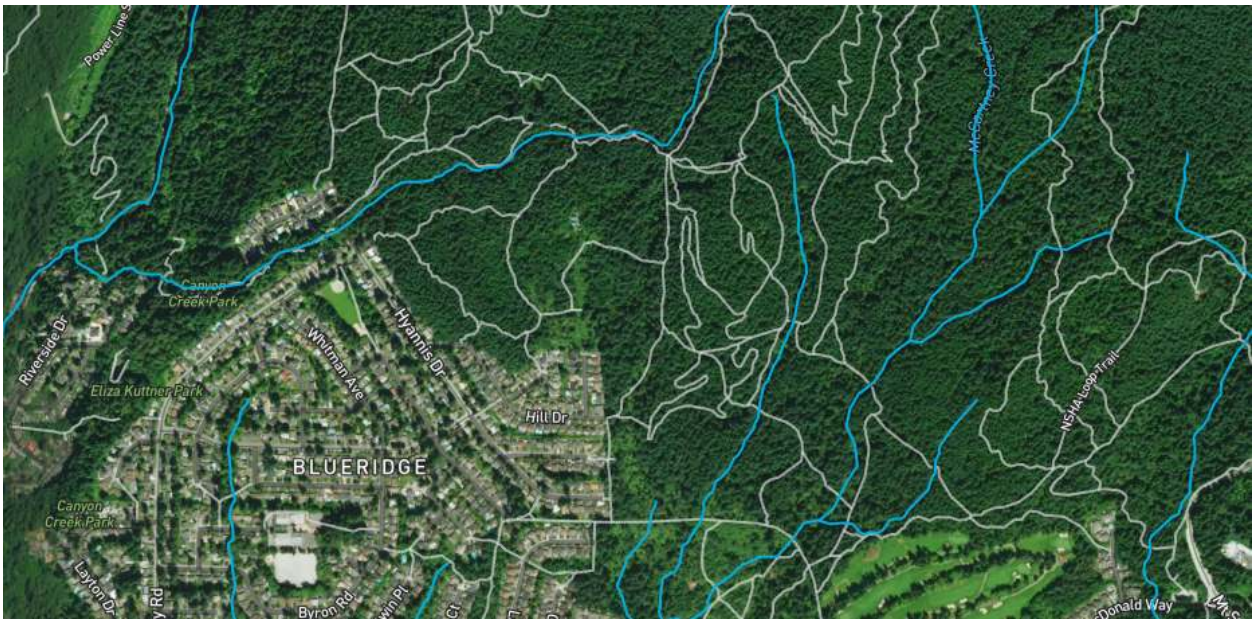
Adding Details to Geospatial Features

For each geospatial feature (e.g. vegetation, pathways and buildings), we create a feature submesh and extrude it. Without additional information, one way to help users identify a feature would be to assign a natural material and color. However, changing the material and color is not sufficient for visualizing certain features, such as vegetation regions. Therefore, we need to add special characteristics and details to feature submeshes. For each feature, we add details to its submesh while considering two important goals: a) how real it looks (at the scale of our physical model), and b) how easy it is to identify. We address these two goals by using aerial photos of the region that we are printing as a reference to evaluate whether the resulting feature submeshes look real, and also by using general-purpose maps to check if the feature types can be identified at least as easily as on those maps.

In this chapter, we discuss adding details for three major feature types in landscape models, namely pathways, vegetation and urban structures. For each of these features, we explain our design decisions for adding details that respect the aforementioned goals.

6.1 Pathways

The first type of feature we consider consists of roads, trails, waterways and similar features that define a pathway on the land (see Figure 6.1). The raw information for this type is a sequence of points (i.e. feature vector), tagged with a number of contextual properties. The available contextual properties vary from region to region. For roads, several common properties are road type, maximum speed, road surface and its name. There is generally no information about the width of a road, though the width is implied by the type of road. Rivers generally do not have width properties tagged to them as well. However, in the case



(a)

(b)

Figure 6.1: Here a number of pathway features are shown. Hiking trails are shown in gray, waterways are shown in blue and red lines represent streets and highways. In (a), a region of Vancouver city is shown, and (b) shows a part of New York city (© Mapbox, © OpenStreetMap).

of big rivers, there is often additional information that shows the boundary of the riverbank as a polygon.

To create and print an extruded layer for this feature type, we create a polygon that resembles a pathway. To do this, we apply an offsetting to the input feature vector to create its feature submesh. If we use the exact width of a pathway for offsetting, it can become too thin at the scale of 3D landscape prints. To help make this feature type more identifiable and create visible pathways when we print, we assign a minimum width to these features (Figure 6.2). This limit is based on the general precision of 3D printers and also the visibility for the target scale of our physical model. Furthermore, we adjust the relative width of different types of pathways based on their type (e.g. a highway road is made wider than a service road). All of the values for the width of each type can be set inside the input file for our system. For more infomration see Appendix A. We also use the Visvalingam and Whyatt curve simplification algorithm [Visvalingam and Whyatt, 1993], to reduce error and increase the reliability of our offsetting. In this algorithm, a given line is simplified by iteratively removing points that result in the least amount of change. There are cases in which pathways have complex interactions, for example in residential areas, that cause assembly of the individual pathways to be difficult. In these cases, we join all of the offset pathways into a single feature submesh and attach them to an underlying layer, as described in Section 6.3, and avoid producing many fragile pathway models.

6.2 Vegetation

The data commonly available for vegetation comes in the form of polygons represented as a set of points. These polygons describe the boundary of the vegetation region, and may be tagged with contextual properties such as tree types and the name of the region. However, these properties are not available for all the regions of the globe.

3D printing vegetation is a challenging task. There are several established methods

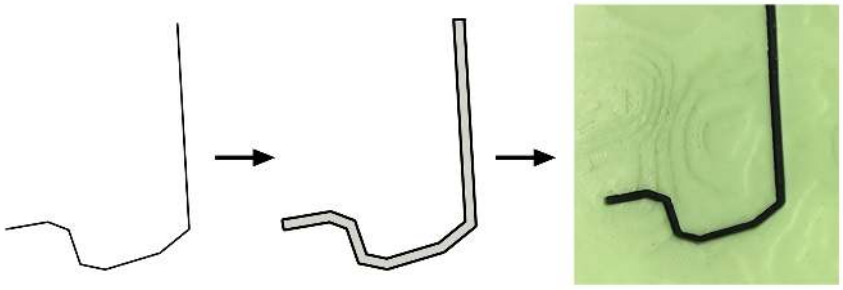
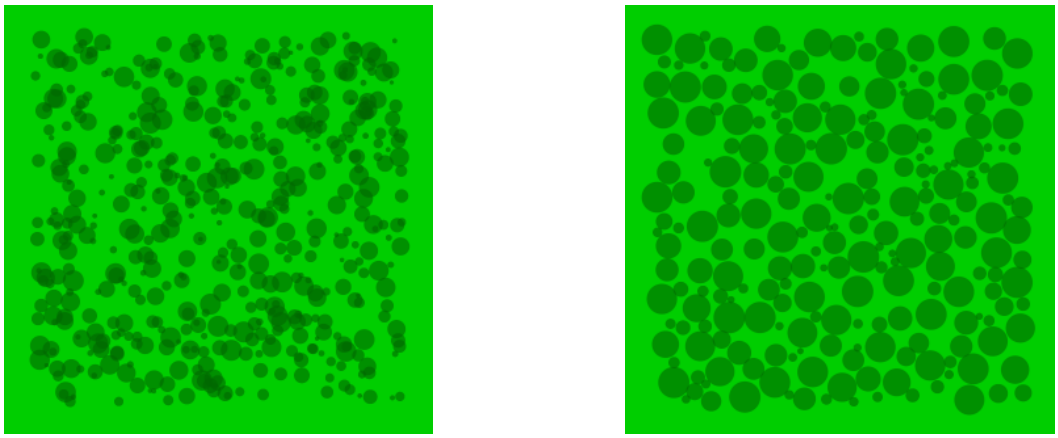
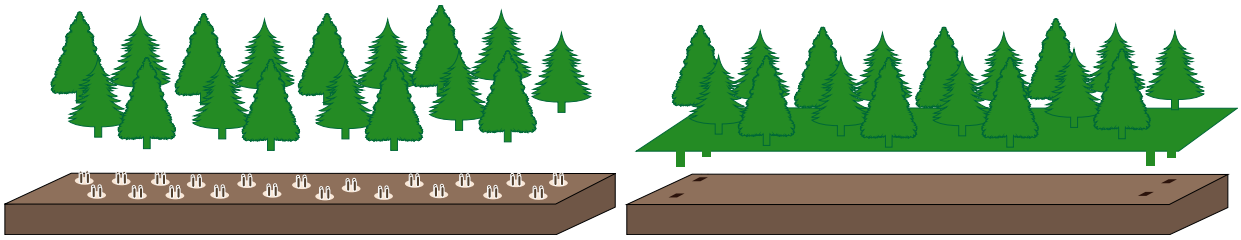


Figure 6.2: We connect offset polylines as a simple and fast approach to create an offset polygon for pathways.



(a) (b)

Figure 6.3: A comparison of random distribution of trees in (a) with our simulation in (b). In (a), there are a lot of overlapping trees, which is not a natural behavior of trees as they compete for access to light.



(a) Individual tree models. (b) Combined tree models.

Figure 6.4: Combining all the tree models into a single layer is beneficial for 3D printing.

for modeling and simulating virtual 2D and 3D vegetation [Deussen et al., 1998, Palubicki et al., 2009, Runions et al., 2007, Ketabchi et al., 2015]. However, for our target scale, printing a huge number of delicate and highly detailed tree models is not possible due to the printer limitations. Also, the miniature scale is fragile and hard to print and assemble. One approach to overcome this issue is to use a small number of large trees to symbolically represent the vegetation area (see Figure 1.1d). With this approach, the final print is more stable. However, this method drastically simplifies the vegetation region by ignoring the density and size of the trees. Therefore, there are two challenges to face: a) populating a vegetation region such that it looks natural, and b) modeling individual trees that are appropriate for 3D printing, yet similar to the real shape of the tree.

To improve the resemblance between the printed region and the real region, we populate the vegetation area with tree models, while creating variations in the size, shape and distribution of trees by using a simulation technique. To find a natural distribution for a vegetation area, we use an L-system grammar inspired by the work of Lane and Prusinkiewicz [Lane and Prusinkiewicz, 2002]. The two main production steps are: a) self-thinning, where larger trees dominate smaller ones and b) senescence, where some of the old trees die and are removed. The final result of this simulation gives a set of tree positions and sizes (Figure 6.3).

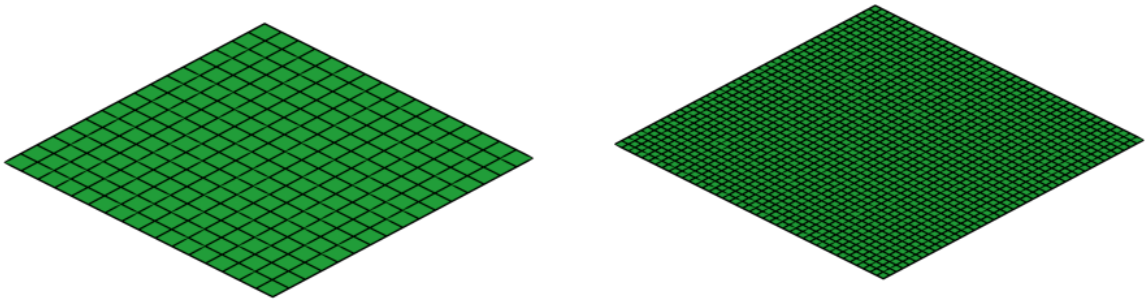
Now, given a natural distribution for the trees, we can create the vegetation area by placing tree models at each of the distributed tree positions and sizing them based on the simulation's result. However, at the scale of our physical models, the vegetation area may

(a) (b) (c)

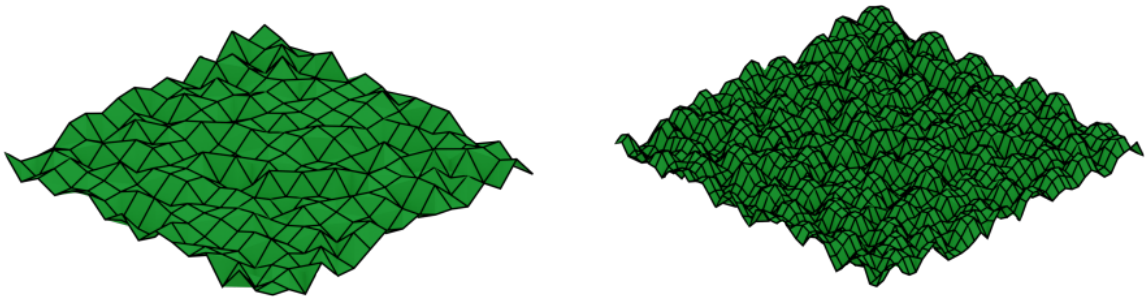
Figure 6.5: Adding tree primitives to the extruded layer. (b) and (c) use displacement mapping to blend the tree primitives in densely populated regions.

cover thousands of trees. For example, our simulation in the Lauterbrunnen region of Switzerland produced more than 5000 tree positions. It is not possible to print and assemble all of these tree models individually. One way to address this challenge is to blend all of the trees on the extruded layer (Figure 6.4).

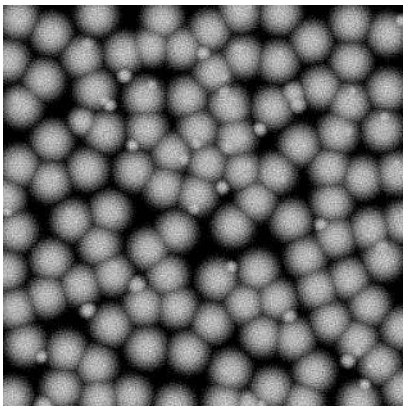
Another issue lies in the complex nature of detailed tree models. The real diameter of a tree in our physical model is too small, and no useful details can be seen at this scale. For example, at the 1:2000 scale of our Lauterbrunnen model, the diameter of each tree on average is about 2 millimeters (based on the average crown width of the Scots pine, which is about 3.8 meters [Sharma et al., 2017]). Therefore, we use simple primitives to represent individual trees. Each primitive is simple for printing, and at the same time resembles a real tree seen from far away. One approach to find such primitives is to simplify the 3D model of a tree and compare it with a set of pre-defined primitives to find the best matching primitive. For example, pine trees can be modeled with cone primitives. These primitives can be sized according to the simulation results and placed on a base extruded layer. In regions that are densely populated with trees, these simple primitives can have many intersections, which are extremely hard to print. For these regions, our system applies a displacement map [Marschner and Shirley, 2016] to the feature submesh, which blends the simple primitives with the underlying mesh (Figure 6.5). Each vertex in the feature



(a)



(b)



(c)

Figure 6.6: In (a), a mesh and a refined version of it are shown. By applying the displacement map in (c), the refined mesh better shows the individual trees (b).

submesh is translated by a vector based on this displacement map. As the resolution of this map is much higher than the feature submesh, our system increases the mesh resolution before applying it (see Figure 6.6). Finally, to create a vegetation layer in our system, we have to choose one of the pre-defined primitive types in our system, along with the number of levels to refine the feature submesh. More information for creating a vegetation layer is provided in Appendix A.

6.3 Urban Structures

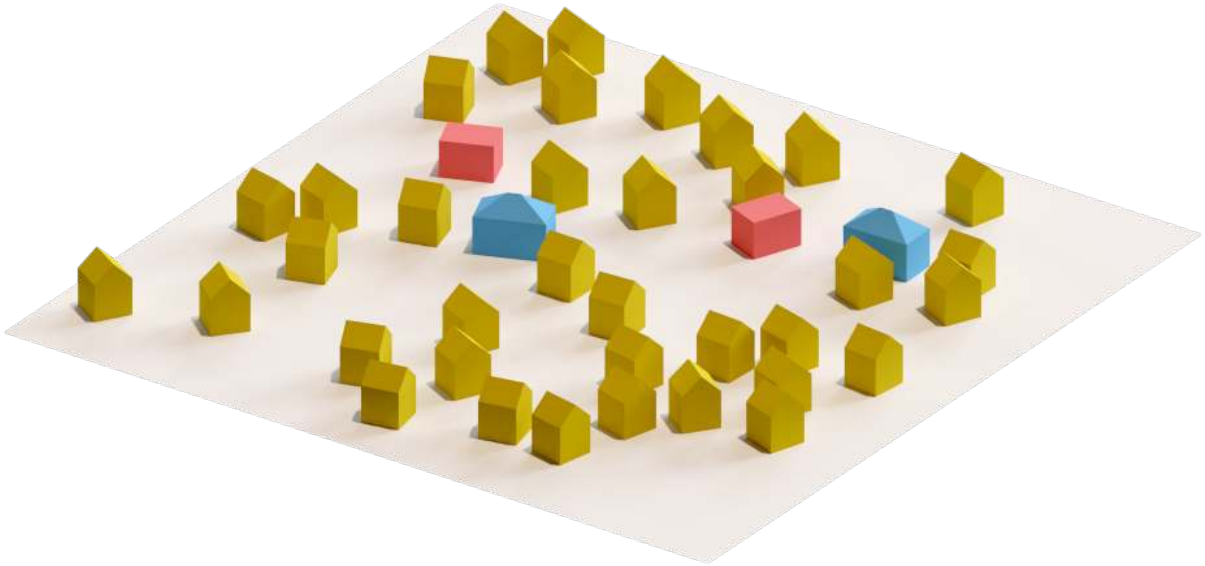
The creation of urban regions faces two main challenges. One challenge lies in creating appropriate models for buildings. Another is that assembling a large number of buildings in such areas will be tedius and time consuming. For many regions in the world, available GIS data lacks detailed information for individual buildings, and the creation of appropriate models becomes difficult. The automatic or semi-automatic creation of building models from satellites and photographs is an active research area and the output of these methods can be used for various applications such as urban planning, virtual tourism and computer gaming. As we only use 2D GIS sources, we employ a statistical approach to creating building models, which is described in the following.

6.3.1 Building Models

Based on the scale of a physical model, buildings can be modeled as simple blocks, individual boxes or as detailed as a house with windows, doors and other features. At our desired scale, roofs play a tangible role in making building models more realistic. The 2D GIS data retrieved by our system only provides the outlines of the buildings and no information about the precise design of each building. We use a statistical distribution approach to randomly design each building's roof based on what can be observed from aerial photos of the region (Figure 6.7). For example, in the photos from Figure 6.7, we may observe that around 90%

(a) Photo by Tom Key (CC BY-SA 3.0)

(b) Photo by Tom Key (CC BY-SA 3.0)



(c)

Figure 6.7: Using aerial photos of a region, we can gather information about how frequently a roof style is used in a region. Based on the images in (a) and (b), we can conclude about 90% of the roofs are gable style, whereas hip and flat style roofs are equally used in the remaining 10% of the buildings. In (c), an example residential region is shown, where this statistical finding is used to design the roof of each building. The gable, hip and flat style roofs are shown in yellow, blue and red respectively.

(a) Gable roof. (b) Hip roof. (c) Flat roof.

Figure 6.8: Different types of generated roofs.

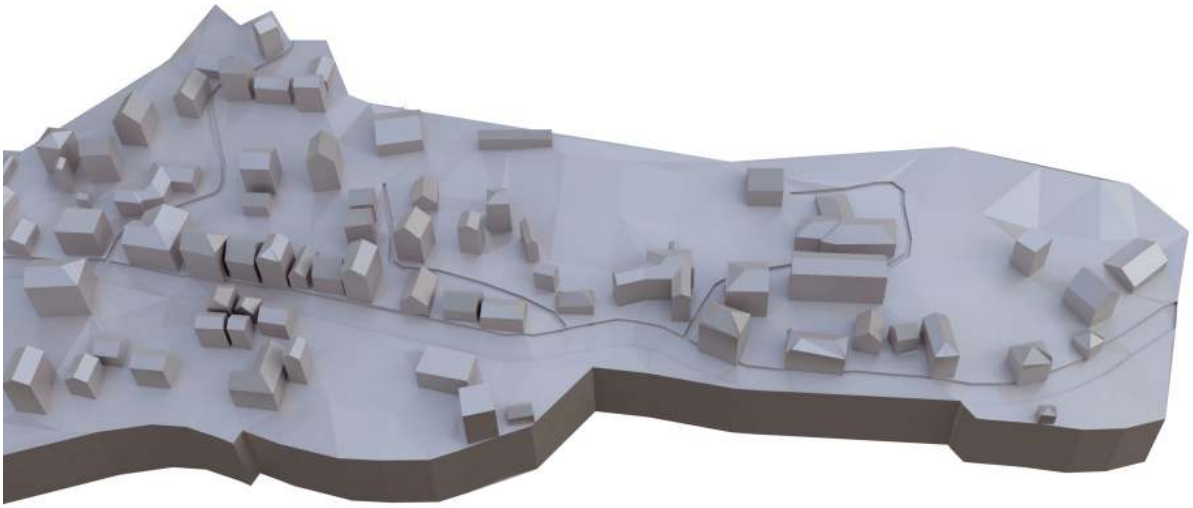
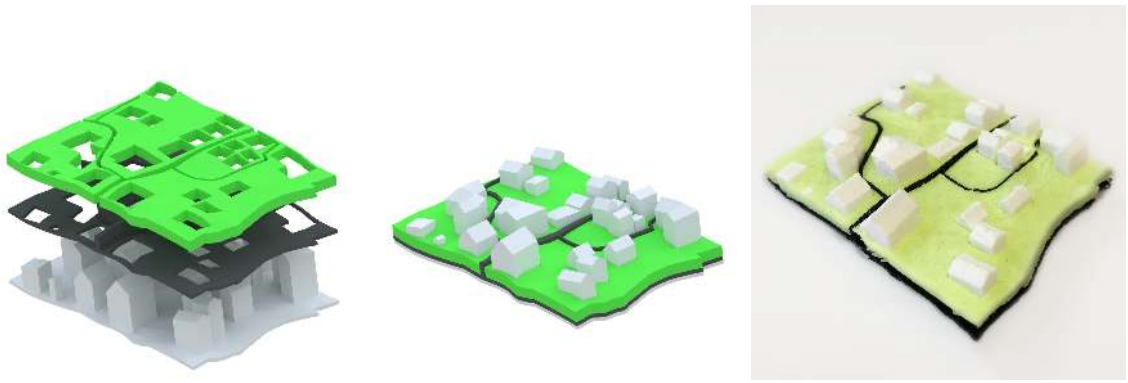


Figure 6.9: 3D render of a set of buildings combined into a single extruded layer

of the buildings have gable style roofs, 5% of them are hip style and the rest are flat roofs (see Figure 6.7c). We then create roofs using these findings and design each roof accordingly (Figure 6.8). To create gable and hip style roofs we extract straight skeletons.

6.3.2 Stencil Layers

To ease assembly of a large number of buildings, we combine all of the building models into a single feature submesh, which results in another 3D element (Figure 6.9). The space between buildings may contain other feature types (e.g. roads and grass fields). To include



(a) (b) (c)

Figure 6.10: We overlay *stencil layers* on top of buildings to add missing and otherwise fragile feature types.

these feature types, our system can generate extra layers to overlay atop the buildings layer (see Figure 6.10). Some feature types like roads can be very delicate and complex to print and overlay. In order to better handle these feature types, our system can combine their extruded layers into a single layer called a *stencil layer*. As shown in Figure 6.10, these stencil layers are created by excluding from the feature submesh those areas through which one should be able to view the underlying extruded layers. In this way, we produce a final result which is still easy to print and assemble, and is consistent with the real world (Figure 6.10c).

Our system is given a configuration file that describes all the features that should be displayed using stencil layers atop a parent layer. In the first step, this parent layer is created by blending a main feature (e.g. buildings) on a base feature layer (e.g. residential area). This main feature is given extra extrusion before blending to compensate for the area that stencil layers will cover. Afterward, each stencil layer is created by creating holes in the base feature layer for the underlying features and blending the corresponding feature details to the stencil layer. Due to the 3D printer's lack of precision, these pieces will not fit together if we print them directly afterwards. Our system applies an offset to the boundary of each stencil layer to help with the fitting.

6.4 Summary

In this chapter, we discussed various methods that our system uses to modify feature submeshes in order to make them both more realistic and more recognizable. For pathways, we have provided an offsetting mechanism to create printable 3D models out of a sequence of points representing the pathways. For vegetation regions, a simulation is run to distribute a large number of trees with varying sizes. Using a displacement map, we provide an easy way to print and assemble massive amounts of trees. Finally, we discussed the creation of stencil layers as a way of printing different features with complex interactions as in urban areas. In the next chapter, we discuss our results along with their accuracy.

Chapter 7

Results and Discussion

The expected output of our system is a physical model that respects the input GIS data. We have tested our system on two regions: Lake Louise in Alberta, Canada and Lauterbrunnen valley in Switzerland. In this chapter, we first discuss the accuracy of our results. We then discuss each of our results in detail.

7.1 Discussion

We checked the accuracy of our physical model by first comparing the 3D model files' dimensions with their respective input GIS data. The error in the 3D model files was much smaller than the 3D printer precision, and were the result of rounding-errors in the floating-point calculations (e.g. intersection and union operations). Our system uses arbitrary-precision arithmetic to vastly reduce these errors, reducing them to a maximum error of around $10^{-13}m$. Practically speaking, the final error for each 3D printed piece is dominated by imprecisions introduced during 3D printing, which makes the assembly difficult.

One source of imprecisions is the 3D printer and the 3D printing settings used. When using FDM process, support structures need to be printed to hold the parts of a model that hang in the air (i.e. overhangs). These parts of the model suffer more inaccuracy due to both the lack of a solid base in those regions and remnants of support structures on the model. Another source of error while using FDM is warpage. As measured and analyzed in [Armillotta et al., 2018], the maximum horizontal dimension of a model can increase this warpage. Therefore, we may conclude that segmenting larger models into smaller pieces could also decrease the imprecisions due to warpage.

By employing vertical extrusion, we prevented overhang imprecisions on the boundary

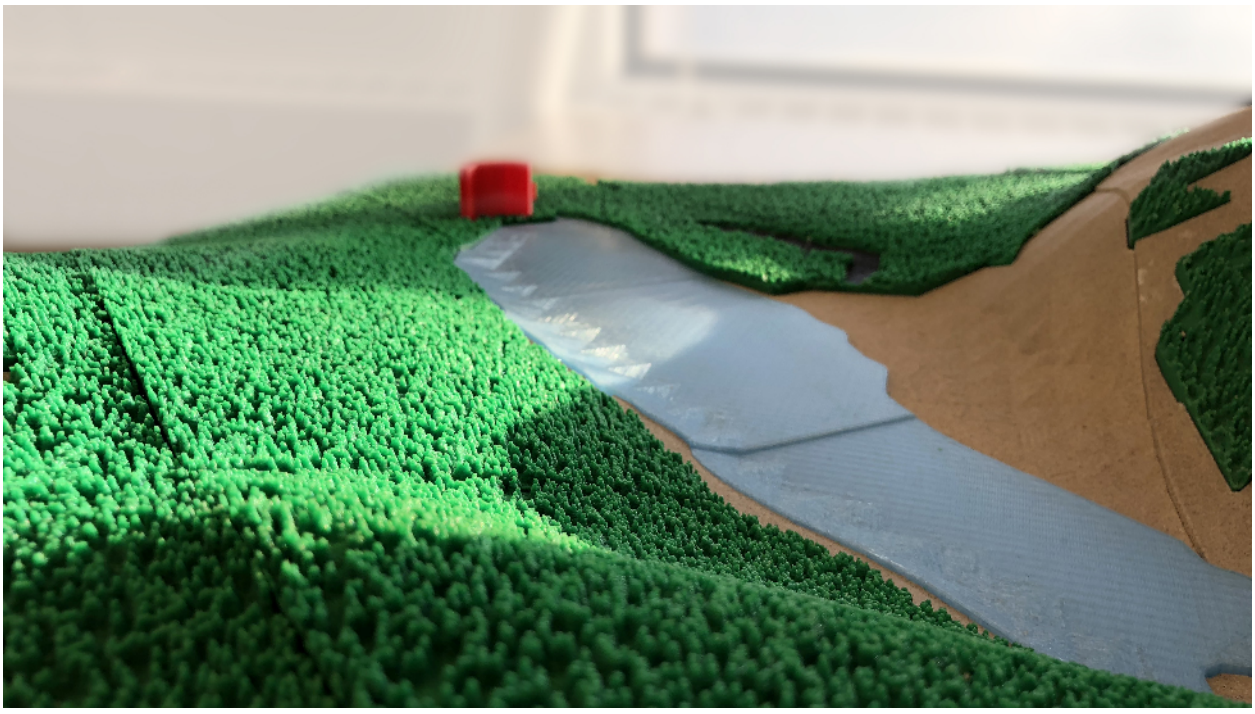
of our pieces. For our first physical model, the Lake Louise model, we managed to fit most of the pieces without any post-processing. However, fitting some parts proved to be difficult and required some sanding. To overcome this issue, after doing several tests, we realized an offset of 0.4 millimeters for the boundary of each piece would make it possible to fit them with no difficulties. Assembling our second physical model, confirmed this offsetting solves the fitting problem.

Based on our experience, the maximum horizontal error for our pieces was about 0.2 millimeters. However, the average vertical error was about 0.5 millimeters, reaching to a maximum of 1 millimeter. The main reason for this increased error, as discussed above, is the need to print support structures to hold the model, while another reason would be warpage of larger pieces. The following sections discuss each of our results in more detail.

7.2 Lake Louise

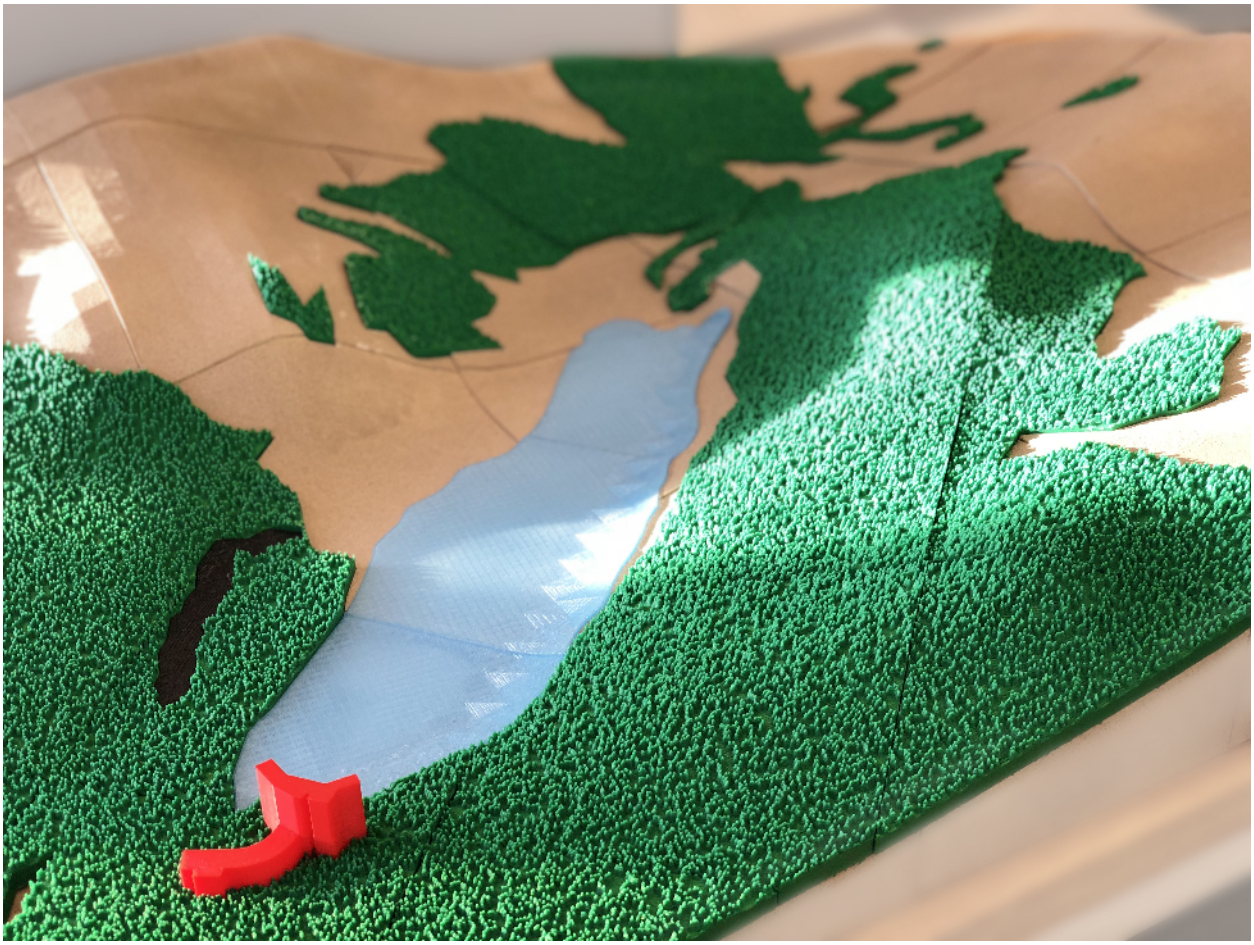
In our first attempt, we used our algorithm to print the area around Lake Louise. The final physical model is $80 \times 66 cm^2$, and consists of 25 pieces for the base model, 23 pieces for the vegetation area and 3 pieces for the lake. Other feature types required no segmentation. The whole model contains 61 pieces that were printed over 55 printing sessions (we achieved 10% reduction in the number of sessions due to packing). A translucent blue was used to print the lake pieces. We used an appropriate color for each of other feature types, for example a green material for the vegetation area. To enhance the aesthetics of the base material, which was printed separately before assembling the model, we applied a textured spray. These separate layers allow us to perform different kinds of post-processing for each piece without affecting the other pieces. The final result can be seen in Figure 7.1.

(a)



(b)

Figure 7.1: The resulting physical model of Lake Louise area. For more images, please refer to supplementary material.

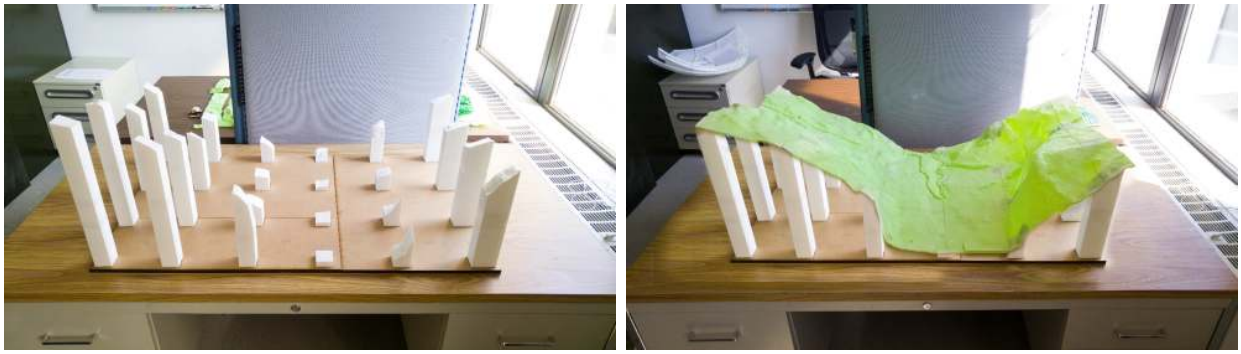


(c)

Figure 7.1: The resulting physical model of Lake Louise area. For more images, please refer to supplementary material.

7.3 Lauterbrunnen

After evaluating the Lake Louise physical model, we realized the available data sets were not diverse enough, as it mostly contained vegetation areas. Therefore, we decided to test our algorithm with datasets representing more complex features. Lauterbrunnen valley contains many different feature types with a very interesting elevation model that makes it one of the deepest valleys in the Alpine chain, where the mountains rise to more than 700 meters on either side. We used a scale of 1:2000 for the Lauterbrunnen model, resulting in a dimension



(a) Pillars.

(b) Pillars with surface attached.

Figure 7.2: We use pillars to reduce printing time and material use.

of $96 \times 57 cm^2$ for this region.

Lauterbrunnen has a very high frequency of large elevation changes (reaching about $38cm$ of elevation change in our model), making it wasteful to print the entire base naively. Such an approach would take about about 20 days and more than 10 Kg of filament for the base pieces, even when using a very low amount of printing infill (such as 5%). Some parts also have a height beyond the printable volume of our 3D printer. In an attempt to solve these issues, we used small pillars to hold the surface of the model (Figure 7.2). The visible part of the base is created as a thin layer (i.e. 0.5 millimeters thick) and attached to the pillars. Printing time for these pillars is far more reasonable, and several pillars can be printed in one printing session.

The total number of separate pieces for this model was about 125 pieces, which were printed over 88 printing sessions with a total printing duration of 29.5 days (we achieved a 30% reduction in the number of sessions due to packing). About 9% of these sessions failed due to the following issues:

- Filament getting stuck in 3D printer nozzle.
- Running out of filament in the middle of a printing session.
- Imprecise leveling of 3D printer's bed leading to weak adhesion of model and

failing the print later.

- Printing over USB and failure of the connection.

As we faced these issues, we tried to reduce them by keeping a precise record of material usage, leveling the 3D printer's bed frequently, and printing directly on the 3D printer using an SD card.

Of the aforementioned printing duration, approximately 14 days were spent on printing the base surface layers and pillars, which shaved a week off of the naive approach and prevented the increased failure rate that accompanies longer printing sessions. Overall, the final physical model used 10.9 Kg of filaments, costing about 327 CAD. The final physical model can be seen in Figure 7.3. One issue we discovered after assembling the final model was a slight vertical misalignment of several adjacent sections (see Figure 7.3b). This is due to stacking several stencil layers. Small imprecisions in each stencil layer accumulate and result in some perceivable misalignments. This issue can be easily addressed in our system by increasing the depth of indentation in the offsetting stage. Figure 7.4 shows a reprint of a small region in the physical model of Lauterbrunnen valley after adjusting this parameter in our system.

7.4 Summary

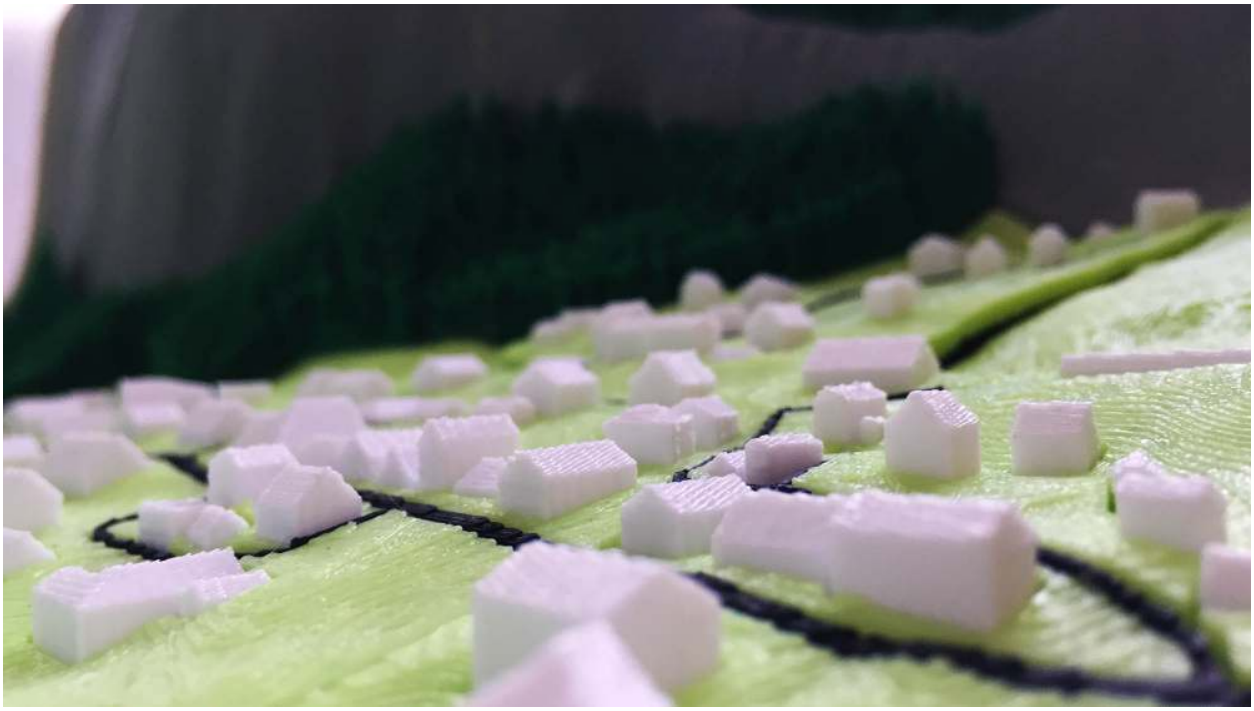
In this Chapter, we discussed the accuracy of our results and the sources of imprecisions. We also discussed our design decisions for reducing such inaccuracies. We gave a detailed explanation of our results along with the challenges faced when creating them. In the next Chapter, we discuss the implementation of our system and a detailed explanation of the components created.

(a)

(b)

Figure 7.3: The resulting physical model of Lauterbrunnen valley.

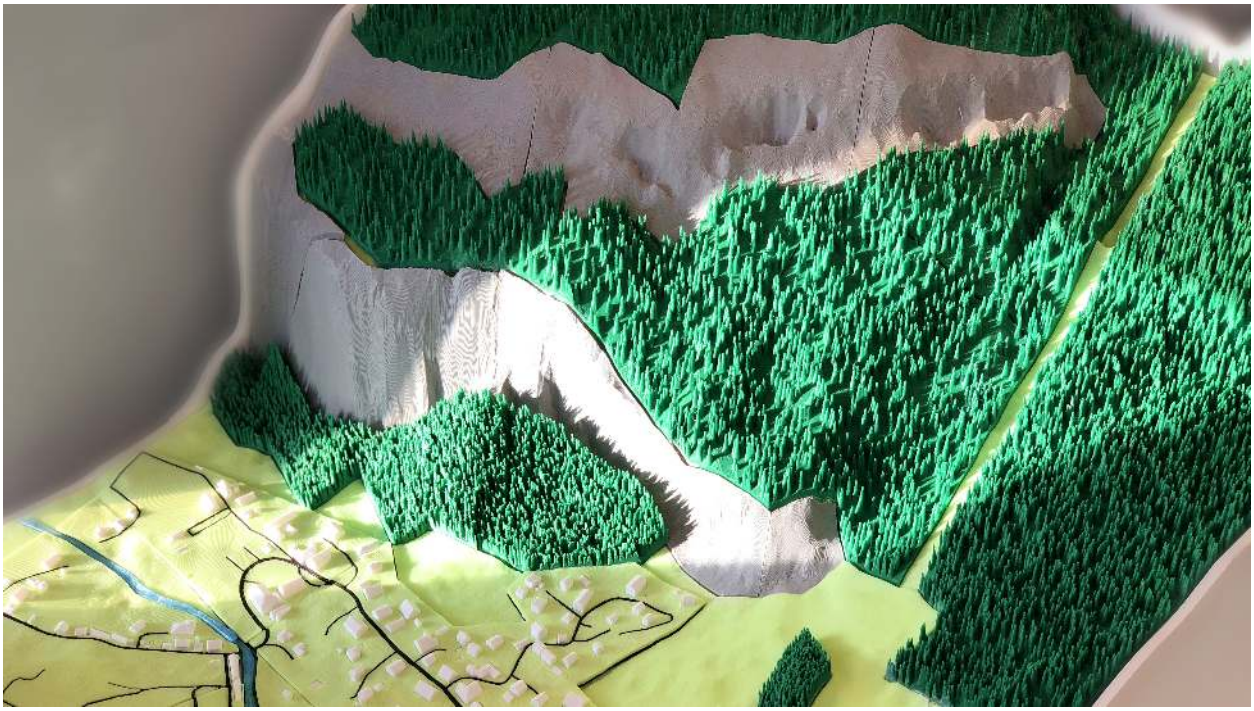
(c)



(d)

Figure 7.3: The resulting physical model of Lauterbrunnen valley.

(e)



(f)

Figure 7.3: The resulting physical model of Lauterbrunnen valley.

Figure 7.4: Using more indentation, our system can account for accumulated imprecision of stencil layers.

Chapter 8

Implementation

We implemented our system in C++ using the CGAL [The CGAL Project, 2018] and Boost libraries [Boost, 2018]. Our system reads an input configuration file and, using the CGAL library, creates an initial CDT. This initial CDT is then processed using different classes responsible for segmentation, adding details to features and exporting 3D models, to create the final printable 3D model files. To avoid loss of precision, we use fixed-precision floating-point arithmetic provided by the MPFR library [MPFR, 2018] through the Boost Multiprecision library. Figure 8.1 shows the class diagram of our system. In our implementation, the `Landscape` class is responsible for reading an input configuration file in JSON format [JSON, 2018]. This file uses a collection of name and value pairs to describe the extents of the desired region to be printed, the required scale of the physical model, the desired features and their properties and the 3D printer maximum printable volume. An example input file can be seen in Listing 8.1. A detailed explanation of this input is explained in Appendix A.

```
{
  "name": "lauterbrunnen",
  "bounds": [7.8954,7.9205, 46.5902,46.6003],
  "features": {
    "meadow": {
      "type": "polygon",
      "filter": "[landuse=meadow]",
      "thickness": 0.5
    },
    "rock": {
      "type": "polygon",
```

```
      "filter": "[natural=bare_rock]",
      "thickness": 0.5
    },
    "forest": {
      "type": "vegetation",
      "filter": "[landuse=forest]",
      "refine_levels": 2,
      "thickness": 0.5,
      "design": "forest",
      "density": 1,
      "shape": {
        "type": "soft_object",
        "scale": 4,
        "max_diameter": 0.096,
        "max_height": 1.2
      }
    },
    "building": {
      "type": "polygon",
      "design": "building",
      "filter": "[building=yes]",
      "z-index": 1,
      "attachment": "combined-layer",
      "order": 1,
      "parent": "residential",
      "orphans": "discard",
      "floor_height": 0.15,
      "floors": [1,3],
      "roofs": {
          "gable": 0.8,
          "hip": 0.1,
          "flat": 0.1
```

```
    },
    "roof_angles": [30, 45],
    "thickness": 0.2
  },
  "residential": {
    "type": "polygon",
    "filter": "[landuse=residential]",
    "fill": {
      "design": "meadow",
      "thickness": 0.3,
      "offset": 0.1
    },
    "thickness": 0.5
  },
  "roads": {
    "type": "union",
    "features": ["highway-secondary","highway-residential","highway-service","highway-unclassified"],
    "z-index": 1,
    "parent": "residential",
    "design": "extrusion",
    "attachment": "combined-layer",
    "orphans": "discard",
    "height": 0.1
  },
  "highway-secondary": {
    "type": "offset_path",
    "filter": "[highway=secondary]",
    "offset": 1.5
  },
  "highway-residential": {
    "type": "offset_path",
```

```
      "filter": "[highway=residential]",
      "offset": 0.8
    },
    "highway-service": {
      "type": "offset_path",
      "filter": "[highway=service]",
      "offset": 0.5
    },
    "highway-unclassified": {
      "type": "offset_path",
      "filter": "[highway=unclassified]",
      "offset": 0.5
    },
    "river": {
      "type": "polygon",
      "filter": "[waterway=riverbank]",
      "thickness": 0.4,
      "z-index": 2
    }
  },
  "print": {
    "scale" : 0.05,
    "search_steps": [5,5],
    "printer_size": [18,18],
    "base_height": 1.2,
    "layer_offset": 0.04,
    "region_offset": 0.005,
    "base_layer_thickness": 1,
    "use_base": true,
    "use_elevation": true,
    "decompose_base": false
  }
```

```
107 }
```

Listing 8.1: An example configuration file for our system.

8.1 Class Roles

In the following sections, we explain the role of each class in our system. Here, we only discuss the most important classes that have a major role.

8.1.1 `Landscape` Class

This class is responsible for reading the input configuration file and creating feature submeshes with the help of the `Triangulation` class. After creating a mesh, this class will also retrieve elevation data and modify the mesh using it. Other classes can access all feature meshes through this class. Other information such as the region boundaries is also accessible in this class. The region information is saved in a class that provides methods to convert between the real-world sizes and the model sizes.

8.1.2 `Triangulation` Class

After reading this file, the `Landscape` class will first create a triangulated mesh using this class. A Constrained Delaunay Triangulation is used to create an initial mesh for the desired region. Each input feature is converted into a set of directed segments and added to this triangulation as constraints. Another constraint on this triangulation is the edge lengths, so that the created mesh will exhibit a high enough resolution for the elevation data. After creating the initial triangulation, using the `mark()` method, we apply our region growing algorithm to find the hierarchy of each feature. This class provides an `iterate_constraint_faces()` method to traverse the faces of each feature.

8.1.3 `MeshExporter` Class

After creating an instance of the `Landscape` class, we can create an instance of `MeshExporter` to save STL files for every feature that we need. This class has a `path` to which the created files will be saved. To export each feature, this class performs two operations: a) extrude the feature and modify its model with a specific design, and b) if the extruded layer does not fit inside the 3D printer, apply segmentation to create smaller pieces. The first task is handled by our `FeatureDesigner` class, and the second is handled using our `Fragmentation` class. This class is also responsible for creating Stencil layers.

8.1.4 `FeatureDesigner` Class

This class modifies the 3D model for each feature based on its type and properties in the configuration file. The simplest design is created using the `create_extrusion()` method, which creates only a simple extrusion with no modification to the surface. The `create_building()` and `create_forest()` methods will modify the surface of a layer to add buildings and vegetation features. Creating vegetation is done with the help of a `Forest` class that is responsible for running a simulation and creating a displacement map.

8.1.5 `Fragmentation` Class

For each feature that is larger than the printable area of 3D printer, this class provides `get_best_fragmentation_mesh_list()` method to obtain a list of meshes that have the minimum value of our cost function. Our cost function evaluation is computed in the `rate_configuration()` method.

8.2 3D Printing

We used a MakerGear M2 as our 3D printer, with a printable volume of $20cm \times 25cm \times 20cm$. Our printer had only one extruder, limiting us to only one material per printing session. For

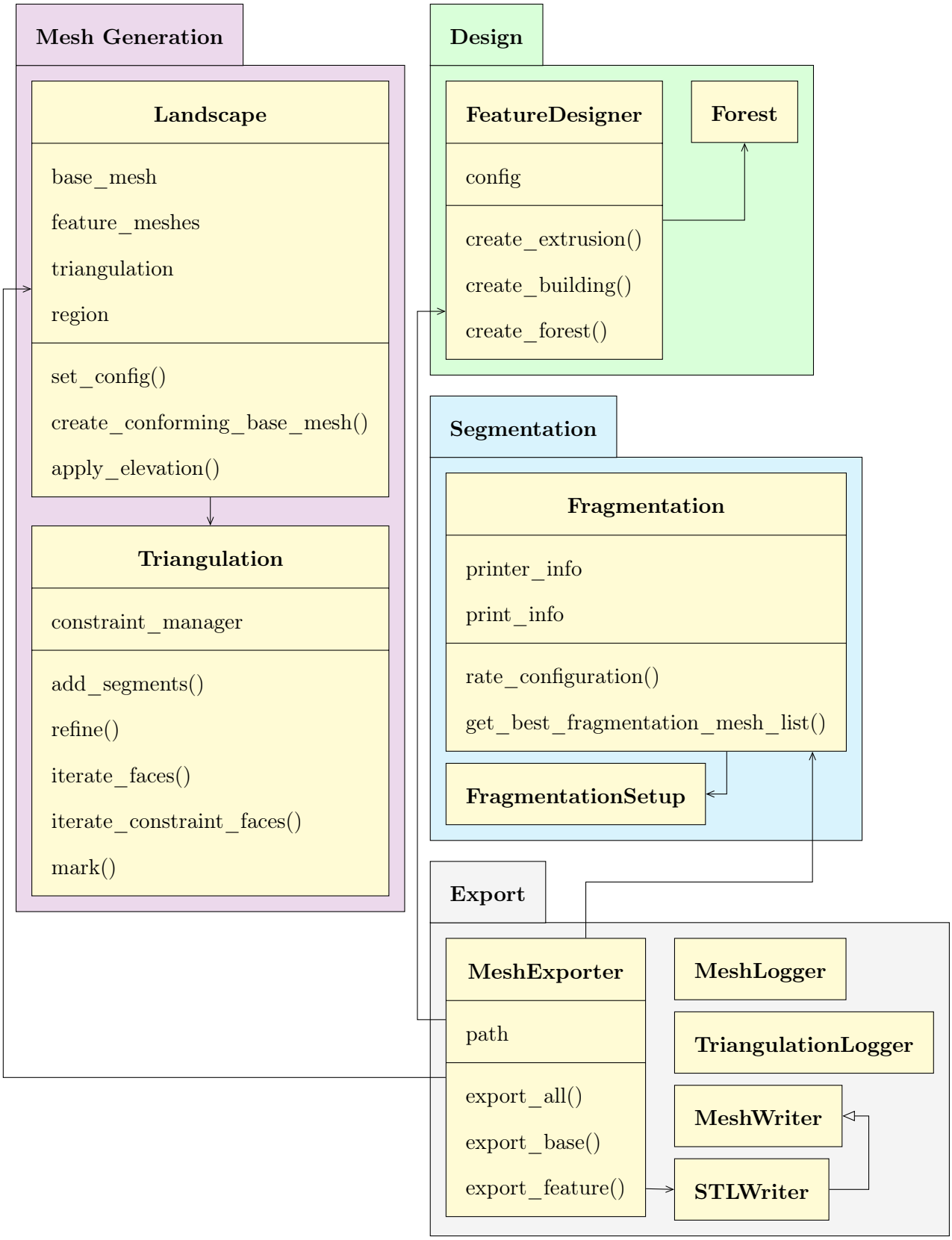


Figure 8.1: Class diagram for our system.

Figure 8.2: Regions shown in red circles show overlapping regions.

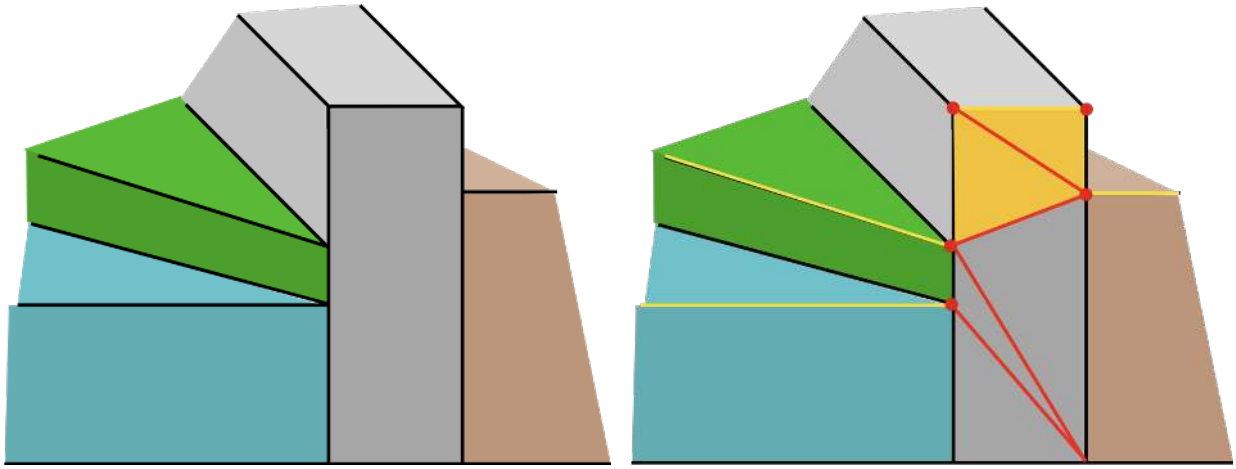
slicing and printing, we used Simplify3D with medium quality settings, and for the printing material we used PLA.

8.3 Technical Challenges

In the following sections, we discuss the technical challenges we faced and how we overcame them. Some of these problems were the result of input data problems, while others resulted from their complexity.

8.3.1 Overlaps

In some cases, feature vector data had overlaps for a single feature. An example can be seen in Figure 8.2. These regions will create unnecessary cuts in our models which become a problem when offsetting. To merge these regions, we check all the input segments for a



(a) A side view of a face of an example base mesh. (b) The resulting triangulation.

Figure 8.3: The different indentation of features requires a proper triangulation. A side view of a face of an example base mesh is shown in (a). The resulting triangulation in our system is shown in (b). The direction of the z-axis is upward in the images.

feature and check if there are two segments with identical endpoints but with a reversed direction. By removing both of these segments, we solve this issue.

8.3.2 Base Layer Triangulation

As different features can have different thicknesses, the base feature needs a proper triangulation to create a valid mesh with connected faces. In Figure 8.3a, a side view of an example base mesh is shown. Each color in this figure shows the corresponding indentation for each feature. The challenge here is to produce correct triangulation of the side faces, such as the gray rectangular face shown in Figure 8.3a, to keep the connectivity of this face with the neighbor faces. To triangulate this face, we find all the intersecting indentations from neighbor faces. The result will be the red dots in Figure 8.3b, which includes the top two points of the face as well. After finding these intersections, we sort them based on their z value for both sides of this face. We start the triangulation by traversing one intersection point of each side, starting from the highest points. Using these two points and the previous

two points (set to the topmost points of the face for the first iteration), we come up with a quadrilateral that can be triangulated by connecting two of the opposite points. When one of the sides runs out of points to traverse, we can triangulate the remaining points by creating fan triangles. There is no need for ear clipping, as the remaining face will be quadrilateral, which is a convex shape. The final result of the example discussed can be seen in Figure 8.3b.

8.3.3 Floating-Point Problems

The most common way to represent a decimal number in a computer is by using a floating point representation. When performing a lot of mathematical calculations, the imprecision of these numbers accumulates and can create problems such as catastrophic cancellation. There are several parts of our system in which these errors can create a problem. Offsetting curves and meshes and clipping feature vectors are examples of algorithms that can fail if we use floating point representation. We use the MPFR library, which provides fixed-precision floating-point arithmetic, in order to solve this. We used 50 bits for the precision of our variables.

8.4 Summary

In this Chapter, we explained how our system was implemented. We provided an overview of the classes implemented and how they work together to create the final printable 3D model files. We also discussed several technical challenges we faced during implementation and how we devised solutions to them. In the next Chapter, we summarize the thesis and discuss possible future works.

Chapter 9

Conclusions and Future Work

We presented a system that can print physical models of landscapes using an affordable 3D printer, and provided solutions for two main challenges that must be faced when printing with these printers. The size limitation of 3D printing is addressed by using a grid segmentation algorithm that is applied locally to each feature submesh. Using separate models for different feature types and creating stencil layers both provides a way to use any number of materials and to incorporate complex feature interactions without producing very delicate feature layers.

Using a square global grid for segmentation makes our method efficient and easy to implement. However, it may result in some undesirable pieces. We apply a simple post-processing step that will merge some of these undesirable pieces. Using non-square grids or more sophisticated post-processing can help to generate more appropriate pieces.

We have provided a method to print vegetation areas using a displacement map. This method can be extended to create different types of vegetation areas with different types of trees and more realistic distributions based on density information from GIS sources. These designs can also be extended to create appropriate textures for other types of features, like mountains, deserts and water bodies.

By creating stencil layers, we provided a way to print several feature types in a region with complex interactions. Even though we used very thin underlying base layers for blending features, the material usage can be further reduced by carefully removing unnecessary unseen parts of these layers.

As our system is designed for the scale of landscape models, fine details of the features, such as windows for buildings or power lines, can not be created with our system. Another

limitation of our work is the cost function we have used does not consider all the aspects of undesirable pieces and may produce parts with shapes that are not strong enough (e.g. a narrow region in a piece).

To expand this work, a user study will be helpful to compare the results of our system with other visualizations and find the important aspects of the visualization that can be improved. Adding more feature types is a possible future work. Examples include 3D printing trains, railways, aerial lifts and bridges. By increasing the target scale of our physical models, we can introduce more details to each of the features, such as buildings and vegetation. In this case, more care should be given to overhangs. We can also expand the stencil layers to create more complex features such as flowers in a botanical garden. For example, by combining one green layer as stems with a red layer as flowers, it is possible to create a garden full of flowers. To enable dynamic visualization through a physical model, another interesting future work could focus on introducing internal pipes to route electricity to different parts of the physical model. For example in [Savage et al., 2014], an animated neon sign is created by adding internal pipes to a 3D print. Adding such functionality can help to visualize changing information, such as weather and traffic information. Integrating 3D modeling tools into our system could also prove beneficial, as it enables the modification of features based on the specific needs of each physical model.

Bibliography

[CDS, 2014] (2014). Geobase canadian digital surface model (cdsm).

[VTP, 2015] (2015). Virtual Terrain Project. `http://vterrain.org/`.

[Allahverdi et al., 2018] Allahverdi, K., Djavaherpour, H., Mahdavi-Amiri, A., and Samavati, F. (2018). Landscaper: A modeling system for 3d printing scale models of landscapes. *Computer Graphics Forum*, 37(3). *Proceedings of Eurovis 2018, to appear,* June 2018.

[Applegate et al., 2012] Applegate, C. S., Laycock, S. D., and Day, A. M. (2012). A sketch-based system for highway design with user-specified regions of influence. *Computers & Graphics*, 36(6):685–695. 2011 Joint Symposium on Computational Aesthetics (CAe), Non-Photorealistic Animation and Rendering (NPAR), and Sketch-Based Interfaces and Modeling (SBIM).

[Armillotta et al., 2018] Armillotta, A., Bellotti, M., and Cavallaro, M. (2018). Warpage of fdm parts: Experimental tests and analytic model. *Robotics and Computer-Integrated Manufacturing*, 50:140–152.

[Bächer et al., 2014] Bächer, M., Whiting, E., Bickel, B., and Sorkine-Hornung, O. (2014). Spin-It: Optimizing moment of inertia for spinnable objects. *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH)*, 33(4):96:1–96:10.

[Boost, 2018] Boost (2018). Boost C++ Libraries. `https://www.boost.org`. Last accessed 2018-04-23.

[Brosz et al., 2008] Brosz, J., Samavati, F. F., and Sousa, M. C. (2008). Terrain synthesis by-example. *Communications in Computer and Information Science: Advances in Computer Graphics and Computer Vision*, 4:58–77.

[Campbell and Wynne, 2011] Campbell, J. B. and Wynne, R. H. (2011). *Introduction to Remote Sensing*. Guilford Publications, 5th edition.

[Chew, 1987] Chew, L. P. (1987). Constrained delaunay triangulations. In *Proceedings of the Third Annual Symposium on Computational Geometry*, SCG '87, pages 215–222, New York, NY, USA. ACM.

[Cozzi and Bagnell, 2013] Cozzi, P. and Bagnell, D. (2013). A WebGL globe rendering pipeline. In *GPU Pro 4: Advanced Rendering Techniques*. A. K. Peters/CRC Press.

[Cozzi and Ring, 2011] Cozzi, P. and Ring, K. (2011). *3D Engine Design for Virtual Globes*. CRC Press, 1st edition.

[Culberson and Reckhow, 1994] Culberson, J. and Reckhow, R. (1994). Covering polygons is hard. *Journal of Algorithms*, 17(1):2 – 44.

[De Floriani and Puppo, 1992] De Floriani, L. and Puppo, E. (1992). An on-line algorithm for constrained delaunay triangulation. *CVGIP: Graphical Models and Image Processing*, 54(4):290–300.

[Deussen et al., 1998] Deussen, O., Hanrahan, P., Lintermann, B., Měch, R., Pharr, M., and Prusinkiewicz, P. (1998). Realistic modeling and rendering of plant ecosystems. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 275–286. ACM.

[Djavaherpour et al., 2017] Djavaherpour, H., Mahdavi-Amiri, A., and Samavati, F. F. (2017). Physical visualization of geospatial datasets. *IEEE Computer Graphics and Applications*, 38(3):61–69.

[Goodchild, 2000] Goodchild, M. F. (2000). Discrete global grids for digital earth. In *International Conference on Discrete Global Grids*. Citeseer.

[Halbert et al., 2017] Halbert, S., Samavati, F., and Runions, A. (2017). Parameter aligned trimmed surfaces. In *Proceedings of Graphics Interface 2017*, GI 2017, pages 90 – 96. Canadian Human-Computer Communications Society / Société canadienne du dialogue humain-machine.

[Hilton and Illingworth, 1997] Hilton, A. and Illingworth, J. (1997). Marching triangles: Delaunay implicit surface triangulation. *University of Surrey*.

[Hormann and Agathos, 2001] Hormann, K. and Agathos, A. (2001). The point in polygon problem for arbitrary polygons. *Computational Geometry*, 20(3):131–144.

[Hull and Willett, 2017] Hull, C. and Willett, W. (2017). Building with data: Architectural models as inspiration for data physicalization. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 1217–1264. ACM.

[JSON, 2018] JSON (2018). JavaScript Object Notation. `https://www.json.org`. Last accessed 2018-04-23.

[Ketabchi et al., 2015] Ketabchi, K., Runions, A., and Samavati, F. F. (2015). 3d maquetter: Sketch-based 3d content modeling for digital earth. In *2015 International Conference on Cyberworlds (CW)*, pages 98–106.

[Kvan et al., 2001] Kvan, T., Gibson, I., and Ling, W. (2001). Rapid prototyping for architectural models.

[Lane and Prusinkiewicz, 2002] Lane, B. and Prusinkiewicz, P. (2002). Generating spatial distributions for multilevel models of plant communities. In *Proceedings of the Graphics Interface 2002 Conference, May 27-29, 2002, Calgary, Alberta, Canada*, pages 69–80.

[Lo et al., 2009] Lo, K.-Y., Fu, C.-W., and Li, H. (2009). 3d polyomino puzzle. *ACM Trans. Graph.*, 28(5):157:1–157:8.

[Longay et al., 2012] Longay, S., Runions, A., Boudon, F., and Prusinkiewicz, P. (2012). Treesketch: interactive procedural modeling of trees on a tablet. In *Proceedings of the international symposium on sketch-based interfaces and modeling*, pages 107–120. Eurographics Association.

[Lorensen and Cline, 1987] Lorensen, W. E. and Cline, H. E. (1987). Marching cubes: A high resolution 3d surface construction algorithm. In *ACM siggraph computer graphics*, volume 21, pages 163–169. ACM.

[Luo et al., 2012] Luo, L., Baran, I., Rusinkiewicz, S., and Matusik, W. (2012). Chopper: partitioning models into 3d-printable parts.

[Mahdavi-Amiri et al., 2015a] Mahdavi-Amiri, A., Alderson, T., and Samavati, F. (2015a). A survey of digital earth. *Comput. Graph.*, 53(PB):95–117.

[Mahdavi-Amiri et al., 2015b] Mahdavi-Amiri, A., Whittingham, P., and Samavati, F. (2015b). Cover-it: An interactive system for covering 3d prints. In *Proceedings of the 41st Graphics Interface Conference*, GI '15, pages 73–80, Toronto, Ont., Canada, Canada. Canadian Information Processing Society.

[Manferdini and Remondino, 2010] Manferdini, A. M. and Remondino, F. (2010). Reality-based 3d modeling, segmentation and web-based visualization. In *Euro-Mediterranean Conference*, pages 110–124. Springer.

[Marschner and Shirley, 2016] Marschner, S. and Shirley, P. (2016). *Fundamentals of Computer Graphics, Fourth Edition*, chapter 11, page 270. A. K. Peters, Ltd., Natick, MA, USA, 4th edition.

[McCrae, 2008] McCrae, J. P. (2008). Sketch-based path design. Master's thesis, University of Toronto.

[MPFR, 2018] MPFR (2018). Multiple Precision Floating-point Reliably. `http://www.mpfr.org`. Last accessed 2018-04-23.

[Musialski et al., 2013] Musialski, P., Wonka, P., Aliaga, D. G., Wimmer, M., van Gool, L., and Purgathofer, W. (2013). A survey of urban reconstruction. *Computer Graphics Forum*, 32(6):146–177.

[Olsen et al., 2011] Olsen, L., Samavati, F., and Jorge, J. (2011). Naturasketch: Modeling from images and natural sketches. *Computer Graphics and Applications, IEEE*, 31(6):24–34.

[Palubicki et al., 2009] Palubicki, W., Horel, K., Longay, S., Runions, A., Lane, B., Měch, R., and Prusinkiewicz, P. (2009). Self-organizing tree models for image synthesis. In *ACM Transactions on Graphics (TOG)*, volume 28, page 58. ACM.

[Panetta et al., 2015] Panetta, J., Zhou, Q., Malomo, L., Pietroni, N., Cignoni, P., and Zorin, D. (2015). Elastic textures for additive fabrication. *ACM Trans. on Graphics - Siggraph 2015*, 34(4):12. Julian Panetta and Quingnan Zhou are Joint first authors.

[Panozzo et al., 2015] Panozzo, D., Diamanti, O., Paris, S., Tarini, M., Sorkine, E., and Sorkine-Hornung, O. (2015). Texture mapping real-world objects with hydrographics. In *Computer Graphics Forum*, volume 34, pages 65–75. Wiley Online Library.

[Pérez et al., 2015] Pérez, J., Thomaszewski, B., Coros, S., Bickel, B., Canabal, J. A., Sumner, R., and Otaduy, M. A. (2015). Design and fabrication of flexible rod meshes. *ACM Trans. Graph.*, 34(4):138:1–138:12.

[Prévost et al., 2013] Prévost, R., Whiting, E., Lefebvre, S., and Sorkine-Hornung, O. (2013). Make It Stand: Balancing shapes for 3D fabrication. *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH)*, 32(4):81:1–81:10.

[Runions et al., 2007] Runions, A., Lane, B., and Prusinkiewicz, P. (2007). Modeling trees with a space colonization algorithm. *NPH*, 7:63–70.

[Samavati and Runions, 2016] Samavati, F. and Runions, A. (2016). Interactive 3d content modeling for digital earth. *The Visual Computer*, pages 1–17.

[Savage et al., 2014] Savage, V., Schmidt, R., Grossman, T., Fitzmaurice, G., and Hartmann, B. (2014). A series of tubes: adding interactivity to 3d prints using internal pipes. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*, pages 3–12. ACM.

[Schöberl, 1997] Schöberl, J. (1997). Netgen an advancing front 2d/3d-mesh generator based on abstract rules. *Computing and visualization in science*, 1(1):41–52.

[Schöning and Heidemann, 2015] Schöning, J. and Heidemann, G. (2015). Interactive 3d modeling. In *Proceedings of the International Conference on Pattern Recognition Applications and Methods*, pages 289–294.

[Schüller et al., 2016] Schüller, C., Panozzo, D., Grundhöfer, A., Zimmer, H., Sorkine, E., and Sorkine-Hornung, O. (2016). Computational thermoforming. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)*, 35(4).

[Schumacher et al., 2015] Schumacher, C., Bickel, B., Rys, J., Marschner, S., Daraio, C., and Gross, M. (2015). Microstructures to control elasticity in 3d printing. *ACM Trans. Graph.*, 34(4).

[Scopigno et al., 2014] Scopigno, R., Cignoni, P., Pietroni, N., Callieri, M., and Dellepiane, M. (2014). Digital Fabrication Technologies for Cultural Heritage (STAR). In Klein, R. and Santos, P., editors, *Eurographics Workshop on Graphics and Cultural Heritage*. The Eurographics Association.

[Sellers et al., 2013] Sellers, G., Obert, J., Cozzi, P., Ring, K., Persson, E., de Vahl, J., and van Waveren, J. M. P. (2013). Rendering massive virtual worlds. In *SIGGRAPH '13: ACM SIGGRAPH 2013 courses*. ACM.

[Sharma et al., 2017] Sharma, R. P., Bílek, L., Vacek, Z., and Vacek, S. (2017). Modelling crown width–diameter relationship for scots pine in the central europe. *Trees*, 31(6):1875–1889.

[Song et al., 2016] Song, P., Deng, B., Wang, Z., Dong, Z., Li, W., Fu, C.-W., and Liu, L. (2016). CofiFab: Coarse-to-fine fabrication of large 3d objects. *ACM Transactions on Graphics (SIGGRAPH 2016)*, 35(4).

[Stava et al., 2012] Stava, O., Vanek, J., Benes, B., Carr, N., and Měch, R. (2012). Stress relief: improving structural strength of 3d printable objects. *ACM Transactions on Graphics (TOG)*, 31(4):48.

[Sun and Salvaggio, 2013] Sun, S. and Salvaggio, C. (2013). Aerial 3d building detection and modeling from airborne lidar point clouds. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 6(3):1440–1449.

[Telea and Jalba, 2011] Telea, A. and Jalba, A. (2011). Voxel-based assessment of printability of 3d shapes. In *Proceedings of the 10th international conference on Mathematical morphology and its applications to image and signal processing*, pages 393–404. Springer-Verlag.

[The CGAL Project, 2018] The CGAL Project (2018). *CGAL User and Reference Manual*. CGAL Editorial Board, 4.11.1 edition.

[Thompson et al., 1985] Thompson, J. F., Warsi, Z. U., and Mastin, C. W. (1985). *Numerical grid generation: foundations and applications*, volume 45. North-holland Amsterdam.

[Vanek et al., 2014a] Vanek, J., Galicia, J. A. G., and Benes, B. (2014a). Clever support: Efficient support structure generation for digital fabrication. In *Computer graphics forum*, volume 33, pages 117–125. Wiley Online Library.

[Vanek et al., 2014b] Vanek, J., Galicia, J. A. G., Benes, B., Mech, R., Carr, N. A., Stava, O., and Miller, G. S. P. (2014b). Packmerger: A 3d print volume optimizer. *Comput. Graph. Forum*, 33(6):322–332.

[Visvalingam and Whyatt, 1993] Visvalingam, M. and Whyatt, J. D. (1993). Line generalisation by repeated elimination of points. *The Cartographic Journal*, 30(1):46–51.

[Wang et al., 2013] Wang, W., Wang, T. Y., Yang, Z., Liu, L., Tong, X., Tong, W., Deng, J., Chen, F., and Liu, X. (2013). Cost-effective printing of 3d objects with skin-frame structures. *ACM Transactions on Graphics (TOG)*, 32(6):177.

[Wang et al., 2008] Wang, Y., Weinacker, H., and Koch, B. (2008). A lidar point cloud based procedure for vertical canopy structure analysis and 3d single tree modelling in forest. *Sensors*, 8(6):3938–3951.

[Xin et al., 2011] Xin, S., Lai, C.-F., Fu, C.-W., Wong, T.-T., He, Y., and Cohen-Or, D. (2011). Making burr puzzles from 3d models. In *ACM Transactions on Graphics (TOG)*, volume 30, page 97. ACM.

[Yang et al., 2015] Yang, Y.-L., Wang, J., and Mitra, N. J. (2015). Reforming shapes for material-aware fabrication. In *Computer Graphics Forum*, volume 34, pages 53–64. Wiley Online Library.

[Yao et al., 2015] Yao, M., Chen, Z., Luo, L., Wang, R., and Wang, H. (2015). Level-set-based partitioning and packing optimization of a printable model. *ACM Trans. Graph.*, 34(6):214:1–214:11.

[Zhang et al., 2015] Zhang, Y., Yin, C., Zheng, C., and Zhou, K. (2015). Computational hydrographic printing. *ACM Transactions on Graphics (TOG)*, 34(4):131.

[Zhou et al., 2007] Zhou, H., Sun, J., Turk, G., and Rehg, J. M. (2007). Terrain synthesis from digital elevation models. *IEEE Transactions on Visualization and Computer Graphics*, 13(4):834–848.

[Zhou et al., 2013] Zhou, Q., Panetta, J., and Zorin, D. (2013). Worst-case structural analysis. *ACM Transactions on Graphics*, 32(4).

Appendix A

User Manual

To operate our system, a configuration file is needed as input. This file is created in JSON format, and should have at least the following keys:

Key	Value	Example
name	A string representing the region that is processed	`lauterbrunnen`
bounds	The boundary of the region, shown as an array of [min longitude, max longitude, min latitude, max latitude]	`[7.8954,7.9205, 46.5902,46.6003]`

Table A.1: Minimum required keys for configuration file.

A.1 Basic Feature Configuration

To add features, a `"features"` key should be added with an object as its value. The keys to this object are identifiers for each feature, and their respective values list that feature's specific properties. In the following example we have added a feature called `"meadow"`:

```
"features": {
  "meadow": {
    "type": "polygon",
    "filter": "[landuse=meadow]",
    "thickness": 0.5,
    "z-index": 1
  }
```

```
}
```

Every feature has a `"type"` property which explains how the input GIS data should be processed. Here you can see a list of possible types:

Type	Description
polygon	The input should be used as it is, e.g. a polygon or multipolygon
offset_path	The system should treat the input as a path and offset it using offset property
union	This feature is created by merging the output of other features

Table A.2: Types of Features

Another property which is shared among the features is `"thickness"` which sets how much the final layer for this feature should be extruded in centimeters. For handling partial coverings, a `"z-index"` key can be set. In case of such conflicts, a region will be resolved to a feature with a higher `"z-index"`.

A.2 Querying Features

To query a specific feature for an identifier, we use the `"filter"` property. The syntax for the value of this property is derived from filter definitions for Overpass QL, which is a language to query Overpass API from OpenStreetMap. Here in Table A.3, we list the basic queries that are common. For a more detailed listing of what is possible, please refer to OpenStreetMap documents.

A.3 3D Print Configurations

There are two types of configurations for 3D printing; one is for the 3D printers, and the other is general 3D model generation configurations. Detailed information can be found in

Filter	Description
[name="*featurename*"]	Finds feature by name, e.g. *featurename="Lake Louise"*
[landuse=meadow]	Finds meadows in the selected region
[landuse=forest]	Searches for forest features
[landuse=residential]	Finds residential regions
[building=yes]	Finds all the buildings
[highway]	Finds all the highways.
[highway=*type*]	Finds all the highways of a specific type. Examples of type: secondary, residential, service and unclassified.
[waterway=riverbank]	Finds rivers specified by their riverbank.
[natural=bare_rock]	Finds rocks and cliffs.

Table A.3: Common Filters

Table A.4.

A.4 Buildings

To create buildings, we set an array that describes the minimum and maximum number of floors of buildings, the height of each floor, and a probability object that specifies how often different types of roofs occur in this region. We also set an array of roof angles to randomly assign to buildings. Detailed information on these properties is listed in Table A.5.

A.5 Vegetation

For vegetation, we set the `"type"` key to `"vegetation"`. Creating vegetation requires a high resolution mesh, as we need to apply a displacement map to it. To specify how many times we need to refine the mesh for this feature, we can set the `"refine_levels"` key to any

Key	Description
scale	The ratio of the model sizes in centimeters to real world sizes in meters.
search_steps	The resolution of local grid segmentation as an array of [steps_x, steps_y], e.g. [5, 5].
printer_size	The maximum printable dimensions of the printer in 2D in centimeters as [max_x, max_y], e.g. [20, 20].
base_height	How much extra height should be added to the base mesh in centimeters.
layer_offset	How much offset should be applied to extruded layers in centimeters.
region_offset	How much offset should be applied to input GIS data to prevent overlaps, in centimeters in model scale.
decompose_base	A boolean to set if we should decompose base parts to layers and surfaces for further post-processing, e.g. creating pillars.
base_layer_thickness	If decomposing the base, specifies how thick the base layers should be in centimeters.
use_base	A boolean to set if we should separate base models. If set to false, features layers are extruded more, so they can stand alone.
use_elevation	A debug boolean to enable/disable elevation data retrieval.

Table A.4: 3D Printing Configurations

number. A good number that does not require a lot of memory is `2`. A detailed description of these properties is listed in Table A.6.

Key	Value
design	Set to `"building"` to enable building creation for this feature.
floors	The minimum and maximum number of floors as an array, e.g. [1, 3].
floor_height	The height of each floor in centimeters.
roof_angles	An array listing all the possible roof angles in degrees, e.g. [15, 30, 45].
roofs	An object describing ratio of roof styles, with keys showing roof style and their value their probability. For example, `{"gable": 0.8, "hip": 0.1, "flat": 0.1}`.

Table A.5: Building Creation Configurations

Key	Value
type	Set to `"vegetation"` to enable vegetation creation for this feature.
refine_levels	Number of times to refine mesh faces.
density	How dense this feature should be. Setting to 1 will fully populate the region.
shape	An object describing the shape of each tree. To set the function for creating the tree, use `"type"` key. Currently, only the `"soft_object"` function is supported in the configuration file. Other functions need to be added programmatically. Other keys include `"scale"` to scale the function output, `"max_diameter"` to set the maximum diameter of each tree and `"max_height"` to set the maximum height for trees.

Table A.6: Creating Vegetation

A.6 Union

If we want to combine a set of different features into a single feature, we can use `"union"` type. This type helps one design each feature separately, but merge the resulting features into a single mesh in the end. An example usage is creating roads with different thickness based on their real width and then combining them. To set the features we want to union we set the `"features"` key to an array of feature identifiers. The features listed in a union type will be discarded themselves and will not create 3D models. See the full example for roads in the following listing:

```
"roads": {
  "type": "union",
  "features": ["highway-secondary","highway-residential","highway-
 service","highway-unclassified"],
  "thickness": 0.1
},
"highway-secondary": {
  "type": "offset_path",
  "filter": "[highway=secondary]",
  "offset": 1.5
},
"highway-residential": {
  "type": "offset_path",
  "filter": "[highway=residential]",
  "offset": 0.8
},
"highway-service": {
  "type": "offset_path",
  "filter": "[highway=service]",
  "offset": 0.5
},
"highway-unclassified": {
```

```
    "type": "offset_path",
    "filter": "[highway=unclassified]",
    "offset": 0.5
  }
```

A.7 Stencil Layers

To create stencil layers, we should set a base layer. For example, we can set a residential region as this base layer. To add stencil layers, a `"parent"` key can be added to a feature with a value of the base layer feature identifier. When setting a parent, there are two ways to attach them to the base layer. We can combine them by creating stencil layers, or just blend them with the surface of the base layer, similar to vegetation layers. To account for these two cases, the `"attachment"` key should be set to one of `"combined-layer"` or `"fixed-to-top"`. To create stencil layers, we use the `"combined-layer"` value.

To set the height of each stencil layer, the `"height"` key can be set, which is similar to the thickness of layers. If the sum of the height of the stencil layers does not completely fill the base layer thickness, a special key can be used to fill the remaining space with a different design. The `"fill"` key with an object value can add this function. The current implementation does not completely handle different fill designs. However, this object can have a specific `"design"` key with a value to supported feature designs. Currently, a simple extrusion will be used for all values of this object.

If a stencil layer has some regions outside the base layer region, there are two ways to handle them. We can create the outside regions as normal extruded layers, or we may discard all those features. The normal behavior is to include all the features. If we want to discard such regions, we can add an `"orphans"` key with a `"discard"` value. This key is useful when dealing with features that are not appropriate for printing individually, such as buildings that individually will take a lot of time to attach.

We can change the order of stencil layers by using the `"order"` key with a number value. Stencil layers with a smaller order value will be created first and larger values will be created on top of them.