University of Bergen
Department of Informatics

Tangible Scalar Fields

Author: Anders Syvertsen

Supervisor: Stefan Bruckner

Co-supervisor: Thomas Trautner

UNIVERSITETET I BERGEN
Det matematisk-naturvitenskapelige fakultet

June, 2022

Abstract

Data Visualization is a field that explores how to most efficiently convey information to the user, most often via visual representations like plots, graphs or glyphs. While this field of research has had great growth within the last couple of years, most of the work has been focused on the visual part of the human visual and auditory system - much less visualization work has been done in regards to the visually impaired.

In this thesis, we will look at some previous methods and techniques for visualizing scalar fields via the sense of touch, and additionally provide two novel approaches to visualize a two-dimensional scalar field. Our first approach creates passive physicalizations from a scalar field in a semi-automatic pipeline by encoding the scalar value and field coordinates as positions in 3D space, which we use to construct a triangular mesh built from hexagonal pillars that can be printed on a 3D printer. We further enhance our mesh by encoding a directional attribute on the pillars, creating a visual encoding of the model orientation and improving upon a readability issue by mirroring the mesh. Our second approach uses a haptic force-feedback device to simulate the feeling of moving across a surface based on the scalar field by replicating three physical forces: the normal force, the friction force and the gravity force. We also further extend our approach by introducing a local encoding of global information about the scalar field via a volume representation build from the scalar field.

Acknowledgements

I would personally like to thank my supervisor Stefan Brucker and my co-supervisor Thomas Trautner for their continued support and guidance throughout this project. It is safe to assume that I could not have done this without all their efforts.

I would also like to give an additional thanks to Stefan Brucker for letting me borrow his Novint Falcon, and to the informatics student council of the University of Bergen for letting me borrow their 3D printer. This project would have been quite difficult without them.

A last thanks go to all my friends and family that supported me throughout this project. Their support helped me keep enough motivation to complete this thesis.

Anders Syvertsen
Tuesday 14th June, 2022

Contents

Chapter 1

Introduction

Visualization is a field that handles how to efficiently convey data to humans. It is defined by Munzner [1] as “Computer-based visualization systems provide visual representations of datasets designed to help people carry out tasks more effectively”. While other fields within informatics focus on efficient ways for the computer to solve a problem, visualization attempts to instead augment humans’ capabilities of solving a problem.

Traditionally the field of visualization has relied on the visual system as it carries the highest information bandwidth of the human-perceptive system. This means that visualization is largely restricted to people with good eyesight. Unfortunately, according to the World Health Organization, at least 2.2 billion of the world population were affected by vision impairments in 2019 [2]. Due to the high demand for alternative methods of information transfer, a recent focus in the field of visualization has been on multimodal presentations of data to reach a wider audience.

One way for the visually impaired to perceive information is via other senses instead, which is the idea of *sensory substitution*. Sensory substitution augments the lack of one sense by substituting it with input from another one, for instance by converting written text into speech via a text-to-speech system, or by giving the user audio descriptions of traditional visualizations [3].

Another substitution for vision is through the tactile sense of touch. Kane and Bigham [4] attempt to utilize this by supporting visually impaired and blind students through 3D-printed tangible maps. Holloway et al. [5] instead tried to replace tactile maps for the blind with 3D printed models. In their study, they found that the 3D printed models proved more intuitive compared to their tactile counterparts while remaining as effective in task completion [3]. Research on representing data via physical artifacts also suggests that physical representations of data improve both memorability and information retrieval compared to similar visualization techniques on the same data [6–8].

In this thesis, we explore previous techniques within physicalization, the physical variant of visualization, and contribute by presenting two novel approaches to physicalize 2D scalar field data.

1.1 Scalar Field Visualization

A scalar field is data laid out in a grid of cells over one or more dimensions, where each cell contains a single scalar value and each dimension represents a continuous space [1]. Since all dimensions in a scalar field are continuous, scalar field data is usually sampled or generated from spatial or temporal domains. Some examples of scalar field data can be luminosity of pixels in a 2D picture, air pressure in a 3D volume, fluid flow velocity in a 3D volume over time (3 spatial + 1 temporal dimension) or heights in a 2D geographic map.

When dealing with scalar field data the datasets can become quite big or complex, especially in cases of sampled and simulated multi-dimensional scientific data. Data analysis of big data sets is a tedious task for humans to accomplish alone and computational tools are often required to solve analytical tasks. For smaller datasets, data can be simplified or aggregated using classical statistical methods, but there are cases where statistical data summaries may lose important information in the data presented. A popular example of this is *Anscombe's quartet*, shown in Figure 1.1. Anscombe's quartet presents four sets of data that under normal statistical analysis methods (mean, variance and linear regression) looks to be identical sets of data. But just from plotting the data, as done in Figure 1.1, a human can easily determine that they are, in fact, different sets of data.

Scalar field visualization is an attempt to help humans analyze scalar field data by utilizing the human cognitive and perceptual system's natural ability to process multiple sources of data simultaneously, typically via visual cues to highlight patterns and trends, or via aggregation or projection methods to simplify complex datasets.

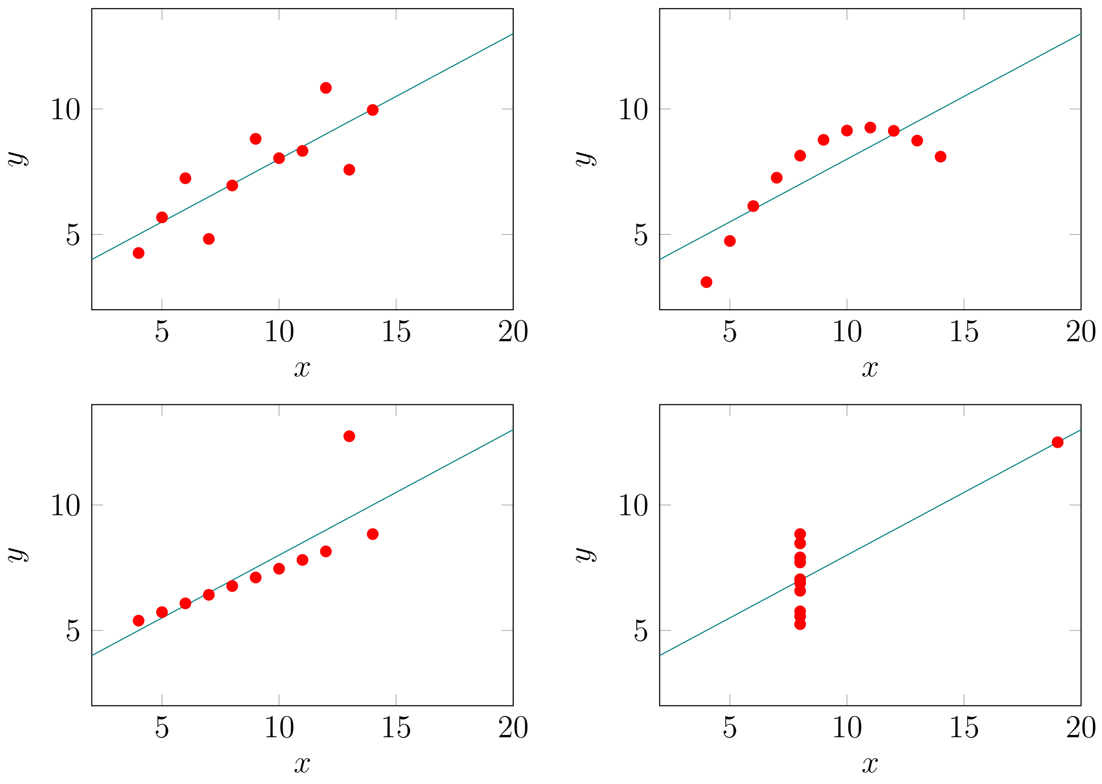


Figure 1.1: Anscombe's quartet is a collection of four datasets that share the same descriptive statistics, although the dataset visually varies greatly. For all datasets: the mean of x is 9, the mean of y is 7.5, the variance of x is 11, the variance of y is 4.125 and the linear regression line (shown in teal) is $y = 3 + 0.5x$

1.2 Physicalization

Physicalization is the act of encoding data through physical artifacts' geometrical or material properties [9] and is a subbranch of visualization. Multimodal visualizations give information to the user via multiple sensory simultaneous channels. But multiple separate channels of information carry with them the issue of connecting the information channels, as incoherent multimodal visualizations only create more confusion. Compared to multimodal visualizations approaches, physicalizations follow an intermodal approach which ensures the multisensory experiences are cohesive, as multisensory outputs stem from the artifact itself [9].

Physicalizations can be *passive* or *active* depending on to what degree of computational power they require after fabrication [8]. Passive physicalizations are completely disconnected from computational machines after fabrication. If the data or representation is changed, another physicalization typically has to be fabricated to reflect the data. On the other hand, active physicalizations can dynamically change what data they represent but are usually still locked to one representation format and usually carry other limitations.

Physicalizations bridge the gap between artistic and pragmatic visualization but has been traditionally difficult to manufacture. Designing efficient physicalizations requires expertise in both visualization and physical fabrication, which highlights the need for tools to automate the process of creating data physicalizations [8, 10]. Swaminathan et al. [10] presented *MakerVis* which is one of the first tools created to automate the physicalization process. MakerVis greatly simplifies the process of manufacturing passive physicalizations. However, MakerVis focuses on abstract data visualizations and manufacturing using cutting techniques like laser cutters or CNC machines. Munzner [1] argues that visualizations should be designed in a task-oriented way. Physicalization solutions should also be designed around the encoding medium [9, 11], meaning there is still plenty of room for more passive physicalization pipelines.

1.3 3D Printing

MakerVis employed cutting techniques like laser cutting or CNC milling, which are classified as *subtractive* manufacturing. However, passive physicalizations are not limited to subtractive manufacturing and physicalizations are also made using the opposite, Additive Manufacturing (AM). AM, commonly referred to as *3D printing*, fabricates models by adding material layer by layer, usually from the bottom up [10, 12]. Compared to subtractive manufacturing, additive manufacturing machines have fewer material options and are usually slower, but can create more advanced geometry and are generally cheaper.

The cheap price of today's AM machines, combined with the innately tactile nature of exploring a physical object, makes passive physicalization via 3D printers an accessible way for the visually impaired to explore data. Due to this, we want to look into creating an automated pipeline for creating passive physicalizations of scalar fields.

1.4 Haptic Devices

Passive physicalizations are limited by their fabrication time and geometry requirements. For instance, separated or flat geometry is difficult to represent as a physical artifact, which puts restrictions on the possible encodings of a passive physicalization. Active physicalizations attempt to work around some of these issues by dynamically changing the representation based on the input data, filters and more. Active physicalization also carries with it the advantages interactive visualizations have over static visualizations. Specifically, it can utilize a constant feedback loop between the user and the output device, which enables interactive visual analysis solutions via active physicalizations.

Active physicalizations based on tactile senses can be done with the help of haptic devices. Haptic devices are machines that can give tactile feedback to a user via vibrations or forces. Haptic force feedback devices, for instance, are devices that deliver feedback to a user through the use of force applied through motors [13, ch. 5.7]. Haptic force feedback devices have been commonly used in the training of medical personnel for needle insertions [13–16], but haptic devices have historically been expensive or difficult to obtain for the common consumer. This changed with 3D Systems developing their popular Phantom haptic force-feedback lineup, attainable by consumers and research facilities. Based on the success of the Phantom devices, *Novint* also released their own haptic force feedback device called the *Falcon* aimed at being the first haptic device available to the average consumer.

With the advantages active physicalizations have over passive physicalizations, we also want to look into physicalizing scalar field data using a force-feedback haptic device. In this thesis, we use the Falcon, but the technique is not limited to this device and should work for any haptic force feedback device with 3 or more degrees of freedom.

1.5 Our Contribution

The main focus of this thesis is on the passive and active physicalization or scalar fields. We first go into detail on current and related techniques in scalar fields visualization, physicalization, additive manufacturing and haptics, after which we will present two novel approaches to physicalize scalar fields.

Our first approach attempts to create a passive physicalization or 2D scalar field data using AM. In this approach, we take a scalar field as input and construct a mesh representing the scalar field, which is then printable by a 3D printer. The geometry for the mesh is constructed as a grid of hexagons representing the field values, which are translated according to each data point's value, after which we mirror the grid to represent the bottom of the mesh and connect the mesh together.

Our second approach attempts to actively physicalize a 2D scalar field by representing it using force on a haptic force feedback device. In this approach, we create surface forces in real-time to mimic a believable surface represented by the scalar field.

Chapter 2

Related Work

Both of our approaches mentioned in Section 1.5 physicalize scalar fields. So in this chapter, we will start by covering some traditional approaches to visualizing scalar fields, before continuing on to the field of physicalization and previous methods for physicalizing data. Finally, we will cover the terminology and the technology of the visualization medium for our two approaches: 3D Printing and Haptic force feedback.

2.1 Scalar Fields Visualization

2.1.1 One-Dimensional Scalar Fields

For simple one-dimensional data, the simplest visualization format is also just encoding each field value along a common axis, which essentially means a point or line chart. In a point or line chart, one axis represents the field keys, and the other axis represents the values. One of the best-known examples of using line charts to represent scalar fields is the *trade-balance time-series chart* by William Playfair, shown in Figure 2.1, in which he visualizes exports and imports between Denmark and Norway, and England over time. Another popular visualization is a bar chart, which shows each field attribute as a vertical bar. Multiple datasets can also be compared in a line chart by using multiple lines or in a scatter chart using different marks for data separation, but another popular representation of multiple one-dimensional scalar fields is *steamgraphs.* Steamgraphs represent a scalar value using area, and multiple datasets can be stacked on top of each other using different color encodings along one common axis.

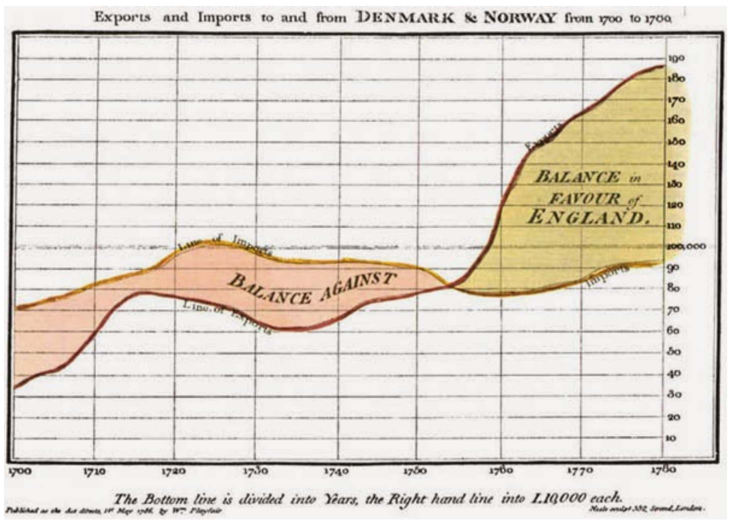


Figure 2.1: William Playfair's trade-balance time-series chart showing the exports and imports between Denmark and Norway, and England over the course of 80 years [17].

2.1.2 Two-Dimensional Scalar Fields

Grid-Based Techniques

For two dimensions, one approach to visualize scalar data is using a *heatmap*. In a heatmap, data values are arranged in a 2D matrix, with each cell value represented with a color. Heatmaps can also additionally encode links between cells in the data using additional glyphs and drawing on the side of the matrix in something called a *cluster heatmap*, which is useful to represent network data [1, ch. 7].

Heatmaps follow a regular grid with each orthogonal axis encoding a new key, but other grid structures also exist. For instance, hexagonal grids have been extensively used in *hexplots* or *hexbin maps*. Like a heatmap, hexbin maps encode a scalar value in a grid, but a hexagonal one instead. An example of a hexbin map can be seen in Figure 2.2 where the number of tweets with "#surf" and their geographic positions are visualized as colored hexagons. The main usages for hexbin maps have been in visualizing binned data. While one also could use square bins to visualize binned data, hexagonal bins are favored for estimation purposes as they give a better estimation bias [18].

Hexbins were first introduced by Carr et al. [20], who additionally suggest that the hexagons can be filled with glyphs to give further information, like how many data points are binned within a hexagon tile. The idea of using a glyph for meta-data is further

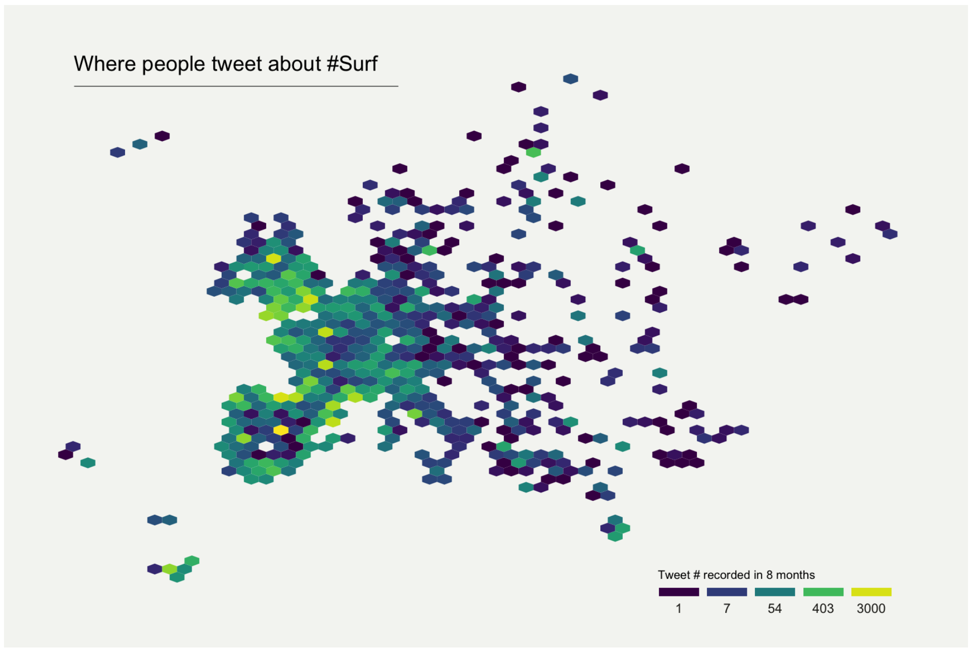


Figure 2.2: Hexbin map visualization showing geographic positions of tweets with "#surf", "#windsurf" and "#kitesurf" [19].

extended by Cleveland and McGill [21], who proposed *sunflower plots*, which are binned scatterplots where each bin is represented by a special glyph. The glyph in sunflower plots displays the binned points as lines from the center of the glyph, somewhat similar to petals on a sunflower. As noted by Carr et al. [20], sunflower plots placed in a hexagonal grid are preferred because they de-emphasize horizontal and vertical patterns that appear in sunflower plots with a square grid layout. Dupont and Plummer [22] later implemented sunflower plots in a hexagonal grid for their Stata platform.

Trautner et al. [23] recently built upon the idea of using glyphs in a hexagonal grid by encoding the distribution of points as a *diamond cut* glyph. Their approach first calculates a regression plane from the point distribution within the bin, after which they construct a hexagonal-pyramid structure that is cut by the regression plane. The cut pyramid is then projected onto a flat surface, which results in the final glyphs represented on each hexagon in their *Honeycomb plot*.

Glyph-Based Methods

Encoding the values of two-dimensional scalar fields along a third axis makes it easier to read off values, but then requires three dimensions of representation space which is typically not available on paper or a computer screen. One approach to solving this is by

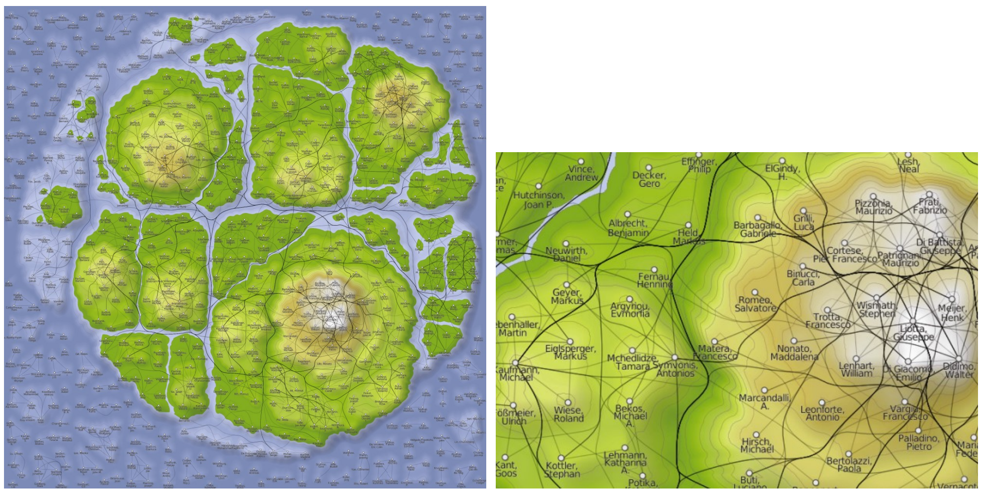


Figure 2.3: In addition to using contour lines, Gronemann and Jünger [24] uses a colormap to encode the scalar value directly, which makes the two-dimensional scalar field look like a topographic map. The right image shows a close-up of the map.

using *contour lines*, commonly also referred to as isolines. Contour lines are polygonal line-glyphs that, per line, pass through all points within a field where a specific value occurs. If the contour lines are uniformly spaced in the value axis, contour lines will show areas of fast change by being close together and areas of slow change by being far apart. Contour lines are commonly used in topographic terrain maps where the contour lines represent the height of the terrain [1, ch. 8].

Topographic maps are commonly drawn with colors indicating different labeled areas of interest, like vegetation, but Gronemann and Jünger [24] instead used a colormap to encode the scalar value, seen in Figure 2.3. An advantage of doing this is that global structures can be encoded via the colormap while local information like steepness at a particular point can be encoded using isoline glyphs [25].

2.1.3 Three-Dimensional Scalar Fields

To represent three dimensions of data on a two-dimensional display, visualization techniques use visual cues like depth, colors and shapes. Most of the three-dimensional scalar field visualizations fall under the two categories: geometry extraction, which are techniques focused on generating and extracting geometric structures representing parts of the data; and direct volume rendering, which instead tries to display the dataset as a whole using computer graphics rendering techniques.

Geometry Extraction

Isosurfaces or contour surfaces are natural extensions of contour lines to higher dimensions. Like a line intersecting through every point with a certain value, an isosurface is a surface intersecting through every point with a certain value.

The first popular approach to visualizing an isosurface was through the *cuberille* technique introduced by Herman and Liu [26]. Inspired by "quadrille" markups on paper, which are obtained using orthogonal sets of equally spaced parallel lines. Herman and Liu [26]' cuberille discretizes 3D into equally sized cubes containing each scalar value in the field, called voxels. For a threshold defined as the surface, if a voxel's scalar value is greater or equal to the threshold, the voxel is rendered and otherwise skipped [27, 28].

Because the cuberille renders in cubes, the final render becomes quite blocky, and if the resolution of the cuberille is too low, smaller values may be aggregated together with neighboring values. To improve upon the surface shape, Lorensen and Cline [29] developed the well-known *marching cubes* algorithm. The marching cubes algorithm makes a finer surface by chaining together sets of polygons created from neighboring voxels. It works by defining cubes where each corner is defined as 8 neighboring voxels and then placing a polygon inside each cube that best distinguishes voxels above and below a threshold [27, 29]. Since each voxel within a cube can either be above or below the threshold, Lorensen and Cline [29] explain that there are $2^8 = 256$ different configurations each cube can have, but also make the observations that all 256 configurations translate to 15 different cases of cubes. It was later observed (first by Dürst [30]) that a few of the proposed 15 cases suffer from *contour ambiguitiy*, which is when multiple polygon setups fit the same configuration of voxels. There are a few solutions that attempt to address this. One of the most widely known attempts at fixing this is the asymptotic decider by Nielson and Hamann [27, 31]. However, Newman and Yi [32] write that most of the issues with contour ambiguity can be avoided by extending the lookup table of polygon cases to 22, which they additionally supply a few examples of.

Rendering a high-resolution marching cubes triangulation can be very costly, so work was further put into making screen-effective algorithms. An early solution was *dividing cubes* which subdivides the voxel space into a grid. The grid size is chosen such that each screen pixel renders exactly one cell in the grid, thus minimizing work put into rendering invisible geometry [33, 34]. Another similar approach was made by Sobierajski et al. [35] called *trimmed voxel lists*, which uses a point within each visible cube and its normal and projects it onto up to three pixels in the image plane in order to ensure that every visible cube is part of the rendered image [34].

Frequently changing the threshold for the surface is expensive, as the surface has to be recalculated. However, only the cubes which contain a voxel above and below the threshold will be part of the generated surface, which allows for acceleration structures. An example of this is the approach by Livnat et al. [36], which uses a kd-tree to quickly determine which cells can potentially be part of the surface [34].

Direct Volume Rendering

Arguably the most popular visualization method for volume data is Direct Volume Rendering (DVR) which directly renders a volume with specific optical properties without extracting any geometric surfaces first [37–39]. During rendering, optical properties are accumulated along a viewing ray to form the final image of the data [39]. Volume rendering is either implemented in an object order approach, which means the volume is sampled in slices which are then blended together, or in an image order approach, which means the volume is directly sampled from screen-space viewing rays using raymarching. The DVR process is divided into three parts: reconstruction, classification, and shading.

The first part, reconstruction, has to happen when the data is sampled at a lower rate than the resolution used to render the data. The domain of the data might be continuous, but sampling a continuous field turns it into discrete measurements, and thus having to reconstruct the in-between values might be necessary.

The classification part controls how the sampled values should be interpreted and what they should identify. For example: in a scalar field representing the densities in a volume, a low-density value likely refers to air and should not be part of the rendered result. Similarly, in a volume of temperature readings, areas with a high value could be rendered with a different color than areas with low values to easily separate them. What value should be treated as what is typically controlled by the *transfer function.* Transfer functions can be a one- or more- dimensional mapping from one scalar value to a color mapping [27]. Kindlmann and Durkin [40] shows how two- or more-dimensional transfer functions using meta-information like gradient magnitude and second derivatives allow for finer tuned classification of elements [39].

The last part is shading, which determines the colors of the classified elements and how they should be blended into the screen. Common lighting models, like Blinn-Phong, are often used to help with the perception of depth and to improve the appearance of objects. Most lighting models use a surface's normal vector to determine how to shade the surface. However, most volumes do not have a defined surface, and the normal is instead approximated using the gradient. In homogenous regions of the volume, the gradient approximation to a normal tends to be inaccurate, so the gradient magnitude is commonly used to determine whether to apply lightning or not [39].

Composition along a viewing ray happens in a *front-to-back* order using the *over* operator or a *back-to-front* order using the *under* operator. In front-to-back compositing, sampling happens in the direction of the viewing ray, starting with the closest sample and progressing away from the viewing origin, while back-to-front composition instead starts with the sample furthest away, moving towards the viewing origin [37, 39, 41]. Modern hardware allows for efficient implementation of image order volume rendering approaches, in which case the front-to-back composition is favored as it allows for early ray termination as an optimization [41].

The appearance of the volume rendering can be improved by implementing shadows. A novel approach to implement *volumetric shadows* can be performed by accumulating light occlusion from the light source within the volume using raymarching and then using the accumulated light to determine the light transmittance at the sample point. The problem with this novel approach is that it requires $O(n_s \cdot n_l)$ operations for n_s samples and n_l light samples, which is quite slow. A better approach is to precalculate a shadow volume whenever possible, which can then be sampled to determine light transmittance. While requiring fewer computations while rendering, this approach instead carries a big memory cost. For image order rendering, Kniss et al. [42, 43] introduced *half-angle slicing*, which enables transmittance calculation simultaneous to rendering. Half-angle slicing works by accumulating light transmittance in slices angled at the halfway vector between the light and the view origin [39].

The Phong-Blinn lighting model implements local illumination, but more realistic results can be achieved by implementing global illumination approximation techniques through light translucence. Light passing through a volume increases and decreases based on energy absorption, emission from other particles in the volume, and light scattering inside the volume. This complicated physical behavior described by the *volume rendering integral* can be approximated using a *phase function*, which approximates light transmittance through anisotropic or isotropic media [44, 45]. The most widely known phase function was introduced by Henyey and Greenstein [46], which describes a simple approximation for anisotropic scattering through a volume. Other popular phase functions are further described by Pharr et al. [44] and Pharr and Humphreys [45].

2.1.4 Scalar Fields in Higher Dimensions

Recent methods have utilized the field of topology to extract geometric features and descriptors to either amplify visualizations or visualize the features and descriptors directly. A lot of recent methods both in extracting topological descriptors and visualizing them are explored by Heine et al. [47] and more recently Yan et al. [48].

Carr et al. [49] introduced a method to efficiently compute the top-level *merge tree* and *contour tree* of a scalar field for any dimensions, where the contour tree is built from the top-level merge tree in linear time. Weber et al. [50] creates a method that takes a contour tree and generates a 2D scalar field from it that describes the surface. They further render the field as a terrain which they call a *topological landscape*. Oesterling et al. [51] build upon this by using the same approach to calculate a 2D scalar field but visualizes it as a color map instead [47].

Fujishiro et al. [52] suggest the use of critical values to guide transfer function designs for 3D scalar field volume renderings. Takahashi et al. [53] further build upon this idea by calculating a contour topology tree which they use to automatically generate a transfer function. Weber et al. [54] use a similar approach in which they calculate the contour tree and use tree branches to define different regions of a 3D scalar field. The regions can then be manually picked by the user to specify a transfer function [47]. Further helping the identification of features, Thomas and Natarajan [55] identify symmetrical substructures within a scalar field by comparing substructures in the contour tree [48].

2.2 Physicalization

Making physical representations of data is not a new approach, but only somewhat recently has work been developed to manifest it as its own research area, known as *Data Physicalization*. Physicalizations are physical artifacts that encode data via its geometrical or material properties [8, 9]. Physicalizations of data have been shown to improve both memorability and information retrieval compared to similar visualization techniques on the same data [6–8]. Compared to multimodal visualizations approaches, physicalizations follow an intermodal approach which ensures the multisensory experiences are cohesive, as multisensory outputs stem from the artifact itself [9].

Augmented physicalizations are passive physicalizations augmented using projections or augmented reality overlays. An example of this was made by Hemment [58] in which

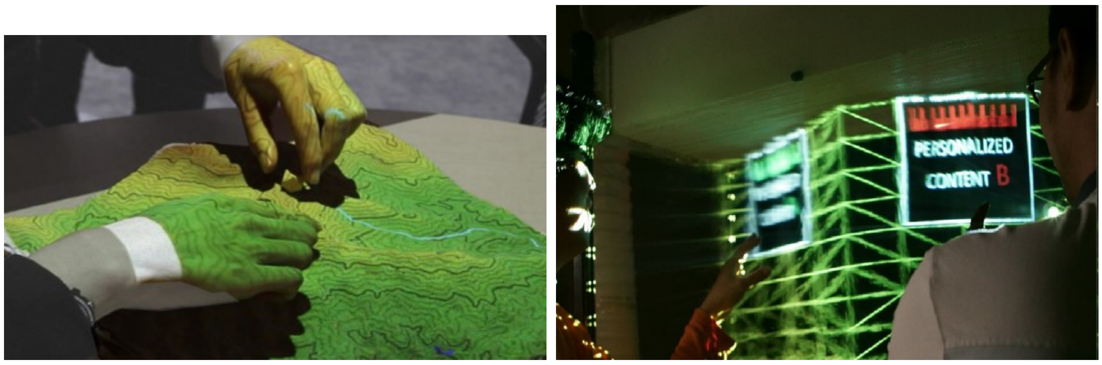


Figure 2.4: The *Tangible Landscape* [56] displays a topographical visualizations projected onto a mallable landscape, reacting to changes in the landscape in real-time. *MistForm* by Tokuda et al. [57] projects visualizations onto a wall of fog that can change its shape.

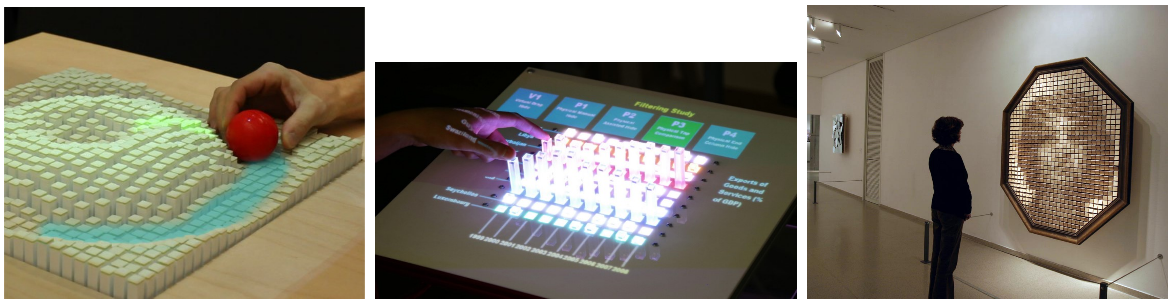


Figure 2.5: From the left to right: Two forms of shape-changing displays using actuated pins; *inFORM* by Follmer et al. [59] and *EMERGE* by Taher et al. [60], and one using rotations; a wooden mechanical mirror by Rozin [61]

they physicalized a height field of Twitter sentiments about the 2012 Olympic Games, which they additionally projected a color map onto to highlight individual stories. Another more interactive example is *Tangible Landscape* by Millar et al. [56], shown in Figure 2.4, which is an educational tool for teaching terrain analysis. Tangible Landscape features a malleable landscape that gets continuously scanned and the topological properties of the scanned surface are projected back onto the landscape [8].

Active physicalizations can update their physical representation based on changes in the data or different encodings. Some examples of active physicalizations are shape-changing displays using arrays of actuated rods or bars, shown in the first two images in Figure 2.5, which can render scalar field data points in a 2.5D fashion by displacing physical rods [9, 59, 60, 62–64]. Shape-changing displays can change their representation in real-time but are limited in their resolution of pins and also cannot render overhangs [8, 65] making it difficult to represent 3D fields. Recently, Nakagaki et al. [66] made an active physicalization via a pin-based shape-changing display where each pin individually can apply force and react to force from the user. Other shape-changing displays able to

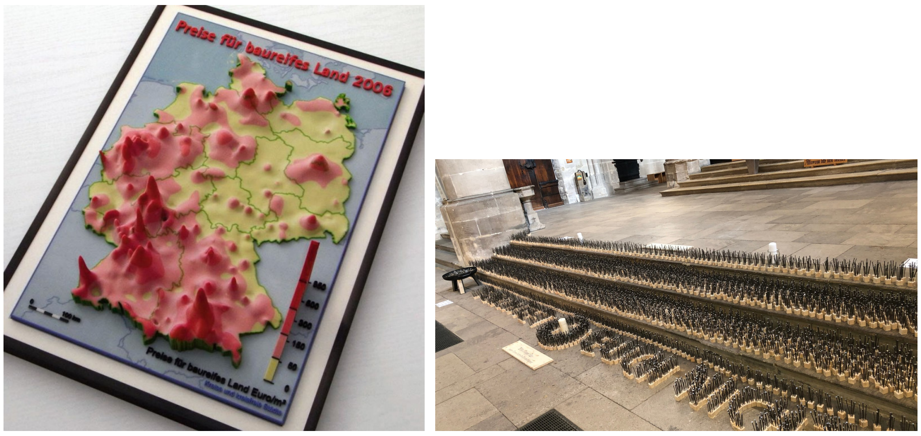


Figure 2.6: From the left to right: A physical elevation model showing average price for building lots in Germany [73] and an art installation in Cathedral of Schwäbisch Gmünd in Germany displaying COVID-19 deaths as nails hammered into wooden cubes [74].

represent scalar 2D field data through texture [67] also exists, for instance using surface displacements [68, 69] or rotations [61, 70], like the one shown in the rightmost image in Figure 2.5.

A few physicalization approaches have explored the idea of using elements suspended by strings or magnetism. For instance, a kinetic sculpture from BMW [71] uses spheres suspended on strings to visualize 2D scalar field data. Omirou et al. [72] visualize similar data by instead using acoustic levitation to suspend elements in the air. Both of these techniques have advantages over shape-changing displays in that they do not occlude as much information caused by pins blocking the view [7], but are still limited to representing one scalar value in their vertical axis. Somewhat in-between an active and an augmented physicalization is *MistForm* by Tokuda et al. [57], shown in Figure 2.4, which projects geometry onto a wall of fog that changes its shape via motors.

Passive physicalizations have been commonly used throughout history, with the earliest known example dating back to 5500 BC in the form of Mesopotamian clay tokens, thought to be used to externalize information [10, 74]. Recent years have shown an increase in data sculptures: artistic physical artifacts which also encode data [8, 10]. For instance, in light of recent events, an art installation at the Cathedral of Schwäbisch Gmünd in Germany, shown in Figure 2.6, displayed COVID-19 deaths as nails hammered into wooden cubes. Nemzer [75] 3D prints a spectrogram of public data on gravitational waves, shown in Figure 2.7, by encoding the spectrogram color as height instead. In comparison to the technique by Nemzer [75], which is also a physicalization of a two-dimensional scalar field, we present a semi-automated pipeline to construct the physicalization whereas Nemzer [75] does not enclose any details on how the physicalization

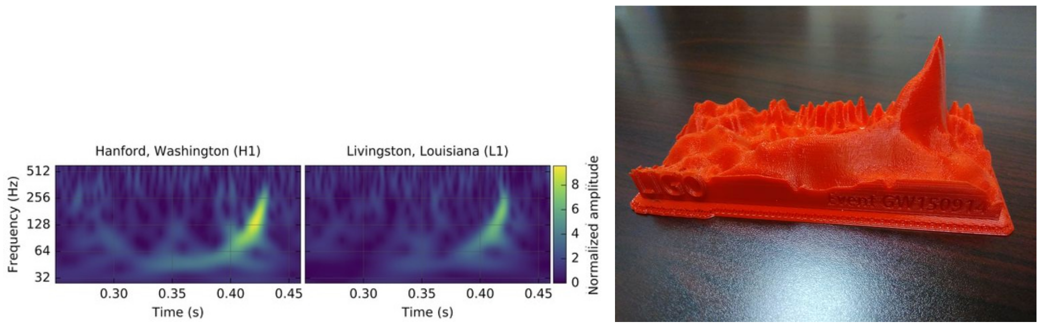


Figure 2.7: Nemzer [75] 3D printed a spectrogram of gravity waves collected from the Laser Interferometer Gravitational Wave Observatory (LIGO).

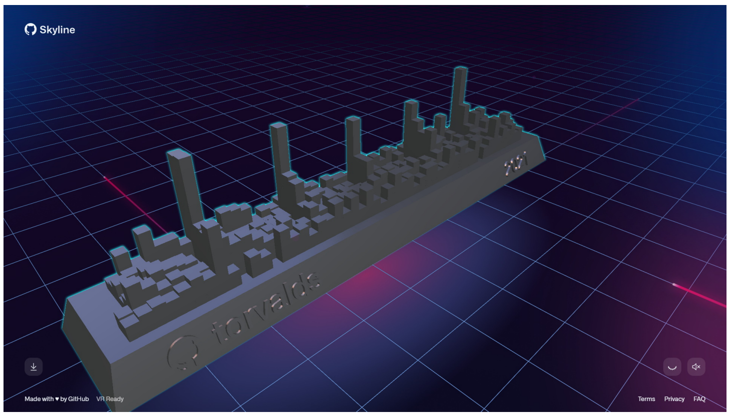


Figure 2.8: GitHub Skyline is an online tool where one can create 3D printable physicalizations of a users commit timeline [76].

was fabricated. GitHub recently made *GitHub Skyline*[76], shown in Figure 2.8, which is an automated pipeline and browser-based tool that creates a printable passive physicalization of a user's GitHub commit frequency by encoding the number of commits per day as heights of pillars in a regular grid. Creating physicalizations with Skyline is a fully automated process requiring only a GitHub username, but the tool is specifically made to represent commit frequency throughout a year, which is a one-dimensional scalar field input but made two-dimensional by letting the second domain represent the week. Skyline's physical representation suffers when tall pillars (created from high field values) are enclosing short pillars and the enclosed pillars' value gets difficult to read, which is an issue we address in our approach (Section 3.4).

2.3 3D Printing

Fabrication of digital models is primarily divided into *subtractive* and *additive* techniques which, respectively, fabricate models by removing or adding material [10, 12]. Subtractive

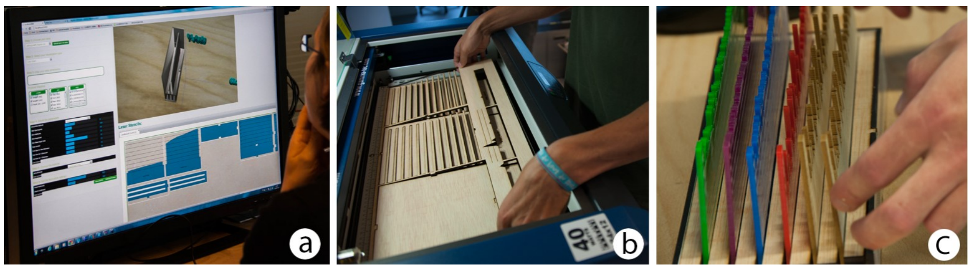


Figure 2.9: *MakerVis* being used to physicalize a bar chart. The design of the physicalization is chosen in the program, before exporting schematics that can be cut by a laser cutter and finally assembling the layers to form the physicalization. [10]

techniques can either operate in 2D, layer by layer, or in 3D like with CNC milling. Some literature further categorizes subtractive layer-based methods as *cutting* techniques. Examples of cutting fabrication techniques include laser cutting, water jets and plasma cutting [8].

Swaminathan et al. [10]' *MakerVis*, shown in Figure 2.9, is an automated process that uses *subtractive* manufacturing to physicalize data. Given an encoding format chosen in the program, *MakerVis* translates the data to be physicalized into layers that can be fabricated using cutting techniques and then manually assembled to complete the physicalization. Umapathi et al. [77] present a possible method to skip the assembling step by utilizing laser cutters both to cut the pieces and weld them together simultanously.

AM is done layer-by-layer, usually from the bottom up. Despite most AM techniques generally being slower and having fewer material options compared to their subtractive counterparts [8], the demand for AM technology has been increasing since the 1990s [78]. AM machines tend to be cheaper, the material cost of AM techniques is usually lower and AM can create complex shapes that are hard to manufacture with subtractive manufacturing, making AM very popular among rapid prototyping [12, 78]. The technology used in AM is divided by the International Organization of Standardization (ISO) into seven categories describing the major technology used: Binder jetting, directed energy deposition, material extrusion, material jetting, powder bed fusion, sheet lamination and vat photopolymerization. An overview of the different technologies and their usages is given by Tofail et al. [12] in their paper on AM.

Today's AM market is predominantly based on material extrusion machines utilizing Fused Filament Fabrication (FFF), which is the popularized name of the trademarked technology Fused Deposition Modeling (FDM) [78]. FDM is a material extrusion process where thermoplastic polymer filament is extruded from a nozzle in horizontal cross-sections, layer by layer. Progressing vertically, the layers are stacked together to fabricate a model [78]. Tofail et al. [12] hint toward the development of FFF being a major contributor towards the inexpensiveness of AM machines.

While a lot of cheap material extrusion AM machines exist, Gardan [78] notes that AM machines using stereolithography are also getting progressively cheaper and easier to obtain. For instance in the form of Selective Laser Melting (SLM) or Selective Laser Sintering (SLS) which are powder bed infusion techniques that use a laser to fuse powder material to solid material on a layer-by-layer basis. Or Stereo Lithography (SLA) which is a related vat photopolymerization technique that builds models layer-by-layer using a photosensitive resin liquid that is selectively solidified using light [78].

2.3.1 Materials

While AM has been immensely popular for rapid prototyping, one drawback of the most common AM technique, FFF, has been that fabrication happens layer by layer, typically in a polymer-based material. Layer-based fabrication means that if an artifact is to use different material types, different colors or different textures, parts of the artifact typically have to be printed separately before being manually assembled. However, because of the popularity of AM, multiple recent techniques have tried to address these issues. A lot of these techniques are covered in detail by García-Collado et al. [79] in their recent report.

Techniques for multiple materials in AM have been created for vat photopolymerization techniques like SLA, which involves switching out the photopolymer liquid throughout printing [79, 80]. Similarly, material extrusion systems like FFF have employed setups with multiple extrusion nozzles, each with its own pushing motors [81]. Khondoker et al. [82] instead developed a technique combining multiple filaments in the same mixing chamber, which enables linear combinations of materials as gradients, so-called functionally graded materials (FGM). Ren et al. [83] constructed a 3D printer that could construct non-linear FGM's using polyurethane material diluted with Al_2O_2. In their technique, an active mixing chamber would automatically calculate the required material combination required to display a certain color [79, 84].

More recently, work has been devoted to enabling colors and multiple materials in material jetting techniques [85, 86]. Further work has also looked into methods to print transparent models [87, 88]. Bader et al. [89] recently used material jetting to print 3D scalar fields by discretizing the field into a voxel grid with alpha values determining physicalization representation. Further, they split the composition into separate material layers which could then be sent to the printer and dithered together to blend the color and transparency. A recent technique by Skylar-Scott et al. [90] called multimaterial multinozzle 3D printing (MM3D) uses Direct Ink Writing (DIW), which is a material extrusion technique based on ink rheology as opposed to thermoplastic polymers, to simultaneously print multiple materials [91].

Some attempts have been made toward adding texture to finished 3D printed models. Mahdavi-Amiri et al. [92] made *Cover-it* which takes 3D geometry from an input model and translates it into multiple patches that can be printed on a normal 2D printer. Finally, the individual patches can be manually attached to the printed model. While requiring more manual work compared to multiple material additive manufacturing techniques, Cover-it allows for the use of any texture to cover the model. As a slightly more automated process, Zhang et al. [93] developed a technique to use hydrographic painting, popularly used to texture car parts, to add colors and textures to a printed model. Their approach unwraps the texture onto the 3D model such that the printed texture will then later wrap around the model without distortion. In another approach, Schüller et al. [94] similarly unwrap a texture but prints it onto a plastic sheet which is thermoformed onto a gypsum mold. Thermoforming puts large stress on a model, something that could typically destroy a 3D printed model, which is why they propose printing a negative mold using AM which can then be used to create the gypsum mold [8].

2.3.2 STL File Format

The STL file was created in 1987 by 3D Systems Inc. for their stereolithography solution. The STL acronym originally stand for stereolithography but has gained the backronym Standard Tesselation Language [95]. The STL format describes surface geometry using triangle primitives and surface normals, via triplets of vertices. From its original introduction, it has remained the industry standard format in AM [95, 96]. While the STL format is simple, it lacks specifications about textures and materials, does not specify units, and also does not enforce surface continuity. Due to this, several proposals to change the standard file format have been made [95–97]. An updated specification of STL using an XML format called STL 2.0 addressed most of the issues with the first format [97]. The more recent version of this file format is known under the name Additive Manufacturing File Format (.AMF). Another popular open standard[1] is the 3D Manufacturing Format (3MF), made as a cooperation between some of the most contributing companies for AM

[1]Found by comparing the supported file formats for 3 of the most popular open-source slicer software, as of 14/06/2022 [98–100]

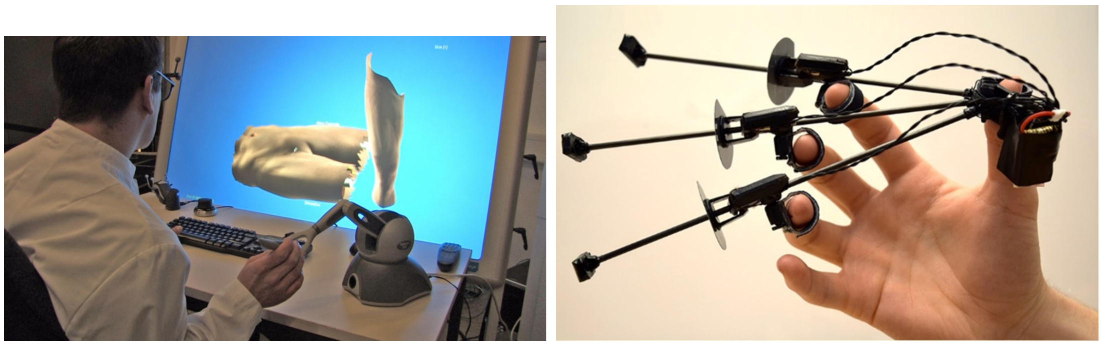


Figure 2.10: From the left: Needle insertion simulation for training personel in applying regional anaesthesia using a Phantom Omni haptic force-feedback device [102], *Wolverine*: a haptic device attached to the hand that simulates the physical restriction of grabbing objects in VR [105].

technology [101]. 3MF can specify material properties for multi-material additive manufacturing but, as Urban et al. [87] point out and propose a solution for, currently lacks specification for translucent materials.

2.4 Haptics Devices

Haptic devices have been most often used in medicine to train personnel [13–16], in which case the device is most often modified with an instrument replica like a needle or a syringe to replicate a surgical environment [13, ch. 5.7]. An example of this can be viewed in the first image in Figure 2.10, where Ullrich et al. [102] use a Phantom Omni to train medical personnel in applying regional anesthesia to patients.

The advances in Virtual Reality (VR) technology also push the focus on haptic research to further improve the experience of VR. Consumer haptic devices designed for VR are gaining popularity, which can be seen from the *TactGlove* being worked on by *bHaptics* which uses linear resonant actuators to give feedback to the user [103]. Currently, most haptics hardware uses linear resonant or voice-coiled actuators which only provide vibration, as they are inexpensive to produce. An exception to that rule is the Reactive Grip by Provancher [104], which is a VR controller that uses actuated sliding plates to mimic force. Choi et al. [105] also explored the use of a hand-attached device that uses brakes on cylinders between the fingers and the thumb to replicate grabbing in VR, which can be seen in Figure 2.10.

Tsai et al. [106] experiments with using rubber balls connected to rubber bands to simulate impact forces on the chest. Chang et al. [107] used belt-tightening motors attached to a VR headset to simulate pressure on a user's face. In a very recent study, Shen et al. [108] developed a technique to haptically render on a user's mouth using ultrasound phased arrays. Ultrasound phased arrays have been increasingly popular for use in mid-air haptics and are even commercially available, e.g. by *Ultraleap* [109, 110]. Weiss et al. [111] tries to use electromagnets to guide the user over a surface using a magnet attached to the user's finger, and Adel et al. [112] tries to utilize the same idea to haptically render 3D geometry in mid-air [110].

Yoshida et al. [113] combined the idea of active physicalizations with wearable haptics and created a handheld pin-based shape display. Koo et al. [114] also tried using dielectric elastomer actuators on a flexible surface that can be wrapped around fingers. However, as Ujitoko et al. [115] writes, both of these methods have limited results in terms of object recognition, which they argue is likely due to the arrangement of pins or actuators not being dense enough. Further, they attempt to address this issue by designing a pin-based shape display for the finger with 128 pins.

Basdogan et al. [116] explore haptic rendering of triangular meshes on a 6DOF haptic force-feedback device with a pen as the end-effector, meaning the physical part the user interacts with on a haptic force-feedback device. In their technique, they utilize the cylinder-like shape of the end-effector to perform collision detection as line and triangle intersections. Fritz and Barner [117] also explore haptic rendering of triangular meshes but using a spherical end-effector instead. In their technique, they check for collision by creating a line between the last and the current position and perform line and triangle intersections. The technique presented in this paper differs in that it haptically renders a scene directly from a two-dimensional scalar field instead of using a triangular mesh intermediate representation.

In their paper, Basdogan et al. [116] also creates textures that can haptically be explored from grayscale images using the computer graphics bump-mapping technique to create the surface normal. To better describe microsurfaces, Choi and Tan [118] attempts to use sinusoidal waveforms and analytically determine the normal. Romano and Kuchenbecker [119] builds upon this by also haptically rendering waveforms, but by using vibration through voice-coiled actuators attached to the cursor device, seen in Figure 2.11. They also made more accurate representations of the surface textures by recording the surfaces with an accelerometer and converting them to frequencies over the Discrete Fourier Transform. The use of vibration makes their technique renderable on modern VR hardware, like the ones by *bHaptics*, due to the common use of linear resonant actuators.

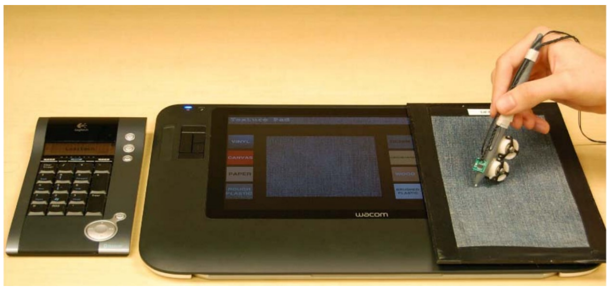


Figure 2.11: Romano and Kuchenbecker [119] simulates textures via vibration on two motors connected to a stylus. The stylus has an accelerometer attached to it such that it can record textures and replicate them using the vibration motors.

2.5 Evaluation Strategies

We will use Kindlmann and Scheidegger [120]' algebraic process to evaluate flaws in the encoding techniques. The process presented by Kindlmann and Scheidegger [120] is a mathematical model to represent common visualization encoding errors, and while both our techniques are physicalizations, Jansen et al. [9] argues that a lot of visualization evaluation and design techniques also apply to physicalizations.

As physicalization is a fairly new research area, not a lot of design guidelines exist. However, Sosa et al. [11] presented physicalization design principles based on the field of industrial design.

Chapter 3

Passive Physicalization

In Section 1.2 we highlighted that while physicalization of data carries with it some positive aspects, a big drawback of physicalizations is the time and skill required to manufacture them. Therefore, a current need in the field of physicalization is the creation of more automated processes for manufacturing physicalizations. In this chapter, we attempt to fill this gap by introducing an approach for manufacturing passive physicalizations from two-dimensional scalar fields in a semi-automated pipeline that outputs geometry that can be fabricated using any AM printer.

3.1 Encoding Format

Similar to a hexplot, we represent our two-dimensional scalar field in a hexagonal grid. As mentioned in Section 2.1.2, hexagonal grids are especially space-efficient when it comes to estimation bias. For 3D printing, hexagonal grids also serve a practical purpose in that they are simple. 3D printing complex structures carries with it the problem of how to balance precision and speed costs. Higher precision printing often requires exponentially more time to perform, which also carries more risks of mechanical or software issues occuring. Using a simple geometry shape for the regular grid means that the whole stucture can be printed with less precision without losing information, which increases speed and printing success rate.

Each AM printer highly varies in terms of printing technique, printing dimensions and other hardware setups. Essentially, to print geometry, each printer requires some middle-ware to convert geometry to instructions for the 3D printer. Traditionally, this is the job of a slicer software. The slicer converts 3D geometry to tool paths for a particular printer, by dividing the volume into slices that can be printed in a layer-by-layer fashion. Slicer software accepts geometry from different file formats, but the most common one is the STL format. This means if we can encode a 3D model in an STL file, most 3D printers, independent of printing technique, should be able to print the 3D model via the help of a slicer.

A study by Fernandez-Vicente et al. [121] also showed that in terms of printing *infill* patterns, which are structural patterns generated on the inside of a printed model by the slicer software, hexagon grids with smaller densities have more structural integrity compared to square grids (rectilinear) with smaller densities. However, as infill density increases, square grids quickly surpass hexagon grids as rectilinear grids can be more efficiently packed to fill the entire volume. As we are more concerned about the structural integrity of the outside grid, hexagon grids could be a more structurally stable layout compared to rectilinear squares.

As discussed in Section 2.3.1, AM is usually constrained to one uniform color for the whole print. While some multi-material AM approaches exist, current multi-material printers are not as common as their single-material counterparts. Unless using multi-material printing, the printed models might lack the color spectrum options to encode enough distinguished values, making the color channel less optimal at encoding the primary attribute of the data in a physicalization. Instead, we can utilize the fact that we are transforming a two-dimensional visualization into three dimensions and encode values along a third axis. Using the position channel makes data visible through haptic sensing, but is also favored for vision because the position channel ranks higher than the color channel in terms of the effectiveness of information transfer [1, ch. 5].

While less useful to users who are fully blind, the choice of colors and materials can still improve readability for users with limited vision. For instance, choosing a glossy material could help the user distinguish or identify the orientation of the surface as rotating and moving the physicalization would reflect light differently based on the surface orientation.

3.2 Geometry Construction

To generate the geometry to print, we first start by constructing a 2D grid of hexagon primitives with each hexagon representing a value in the field, displayed in the first part

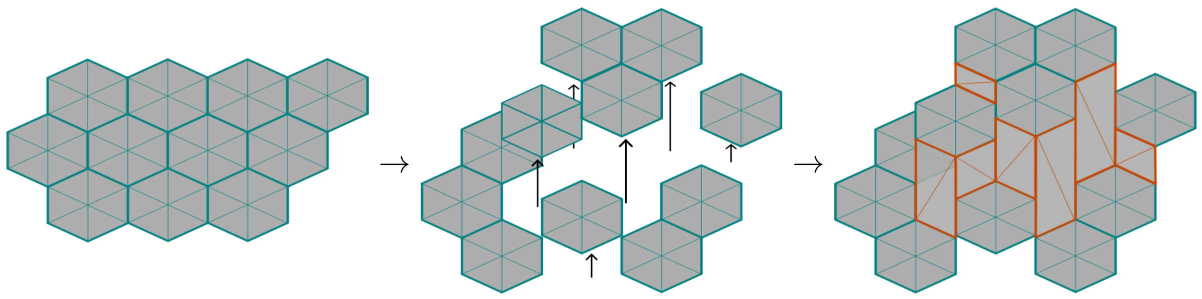


Figure 3.1: The main geometry is created by first a constructing a hexagon grid. Then, each individual hexagon is translated based on its scalar value. Finally, each hexagon is connected to its neighbour via rectangles.

of Figure 3.1. Afterwards, we translate each hexagon primitive along the third axis orthogonal to the 2D grid, with an amount equal to its field value multiplied by a user-defined scaling parameter, seen in the second part of Figure 3.1. The scaling parameter can be adjusted by the end-user to change the resulting printed model's height. This leaves us with a lot of floating unconnected hexagons at various elevations. The STL format requires geometry as a closed surface to be printable, so the next step is to connect them to a connected surface.

In a grid of hexagons, the only thing separating neighboring tiles in the 2D case are edges. In the last step, when the tiles are translated from the ground plane, they are not necessarily translated the same amount, which means the edge separating two hexagons might not be aligned anymore. But as all the tiles are translated along the same axis, the edges between neighboring hexagons remain parallel. All we then have to do to bridge the gap between two edges is to add a simple vertical rectangle between them, which is displayed in the last part of Figure 3.1. Of course, we only have to bridge the gap between edges that are not aligned anymore. So for neighboring hexagons with equal or near equal values, we can simply skip adding a rectangle altogether.

By constructing and extruding a tile for every field value, we include every value in the physicalization, even if the value is zero. While zero values are good for providing a frame of reference, 3D printing zero values wastes precious materials and time where we could have simply excluded the tile instead and used the "ground surface" as the zero value. However, the problem with excluding all zero-valued tiles is that the sets of tiles might be divided into multiple unconnected components, so-called *islands* of geometry, which is illustrated in Figure 3.2. Most 3D printers are fine with printing islands, but the problem with having multiple islands is that the physicalization itself is not consistent within itself. As the horizontal and vertical position between islands is not strictly enforced by being physically connected to the other islands, moving the islands around changes the

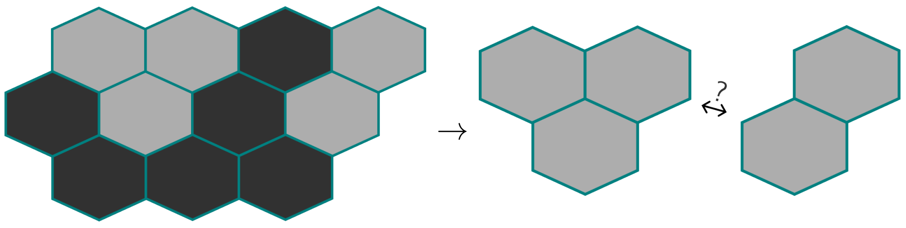


Figure 3.2: Islands problem: Removing all zero-valued tiles creates separate islands which the final 3D print have no way of enforcing relative positions of.

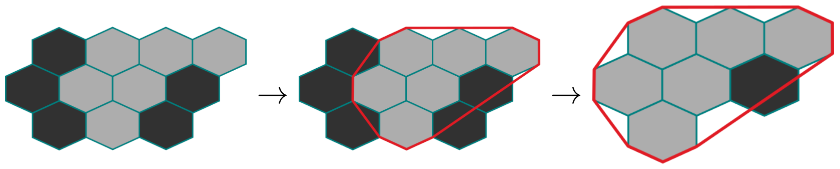


Figure 3.3: The cull is calculated as a convex hull, shown in red, around the non-zero tiles. After the cull has been calculated, hexagons outside the hull are discarded. Hexagons are considered to be inside the hull if their center position is inside the hull, which is why the dark hexagon on the lower right is not discarded.

relative position of islands to one another, indicated by the question mark in Figure 3.2. This means that by interacting with the physicalization, the physicalization now gives the impression that the data it represents has changed. According to Kindlmann and Scheidegger [120], this would be a breach of their principle of representation invariance which states that a visualization should be invariant to the choice of representation. Nothing in the data has changed, but just via interaction it suddenly seems like the physicalization corresponds to a different data set.

To ensure that the geometry is connected while still attempting to save on resources, we calculate a convex hull around all non-zero valued tiles using the well-known *Graham scan* algorithm [122]. Graham scan builds a convex hull around a 2D point cloud by "scanning" through the points in quasilinear time, including points that would make a right turn with the rest of the hull. Any tiles outside the convex hull, which is defined by whether the position of the hexagon is outside the convex hull, get discarded and a single connected island of tiles remains, which can be seen in Figure 3.3.

Calculating a convex hull around the tiles ensures a solid connected surface while decreasing the number of zero-valued tiles we need to include. Compared to including all zero-valued tiles, only including the tiles within the convex hull reduces the required material, thereby reducing manufacturing resource costs and printing time.

Thus far, we have only added geometry between tiles. However, our geometry is not yet printable as the tiles depicting our surface do not specify a thickness and are therefore

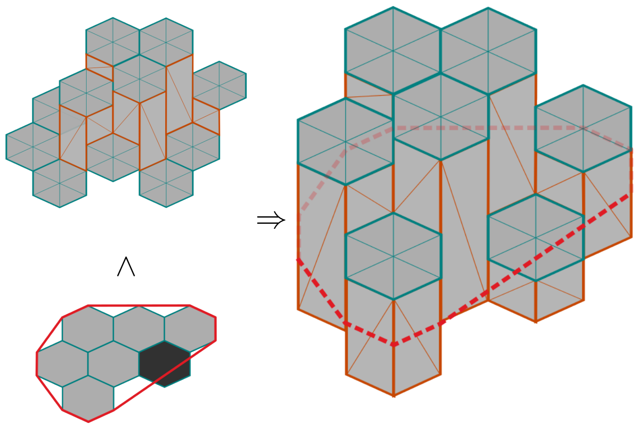


Figure 3.4: After discarding outside hexagons, hexagons inside the hull are extruded down to form the base

completely flat. Furthermore, the far edges of our surface are currently not connected to anything, but the slicer requires a closed surface for the geometry to be printable. Fortunately, since we already calculated a boundary hull around the model, we can use the same boundary hull to create a base for the model which will ensure that our model has a non-zero thickness. To create the base, we start by finding all edges along the convex hull. For each edge, we then duplicate the edge and translate the edge along the opposite axis that was used to translate the tiles before finally bridging the edges together with a rectangle, which can be seen in Figure 3.4.

Constructing a wall around the model has now left us with a single polygonal hole around the boundary at the bottom of the model. Both Guo et al. [123] and Pérez et al. [124] cover a lot of algorithms that can patch missing holes in 3D geometry in their surveys from the same year. However, as pointed out by Guo et al. [123], hole-patching in two dimensions is much easier than doing it in three dimensions and can simply be done using a triangulation algorithm instead. If we then construct the hole by positioning all vertices of the boundary on the same plane, our hole can then be fixed by triangulating the boundary, which most slicer software today also can do for us.

3.3 Directional Attributes

Currently, the top of the pillars (referring to how the top and the bottom tile somewhat resemble a pillar) making up our physicalizations remain untouched, but there is potential to utilize these flat surfaces to encode more data. One such attribute of the pillar is the rotation or tilt of the top of the pillar, which can encode additional directional data. For instance, for a two-dimensional tensor field of three-dimensional values, the tilt of the pillar could represent the direction of the tensor while the height of the pillar could represent the magnitude. As hexplots are often used to visualize aggregated data, we decided to utilize the tilt of the pillars to physicalize the aggregated data's distribution. Specifically, we physicalize the alignment of the point distribution.

In a paper by Trautner et al. [23] they create a visual encoding called a *diamond cut* which they use to display the distribution of points within a hexplot bin. To create their visual encoding they construct a hexagonal pyramid, which is cut in two by a regression plane constructed over the points within a hexagonal bin. The cut pyramid, equalling to the boolean intersection (Constructive Solid Geometry (CSG)) between the regression plane and the pyramid, is then shaded using Blinn-Phong or a similar lightning model and finally projected down onto the tile. Our physicalization, being in three dimensions, allows us to instead directly encode the distribution of points by directly aligning each tile with its regression plane.

Unfortunately, simply rotating the tile to match the regression plane normal, shown in Figure 3.5, would invalidate one of our previous requirements, which is that all the tiles in the grid need to be extruded along the same axis. The edges of the rotated tiles have now shifted horizontally and its corresponding rectangle to its neighbor has to be rotated as well. As seen in Figure 3.5, this can lead to triangular holes in the surface geometry due to how our approach connects hexagons together with rectangles. Instead, we translate every point within a tile along the common translation axis with an amount given by projecting the position onto the regression plane normal. The translation amount is given by Equation 3.1; where h is the translation amount, p is the local position within a tile, $\hat{n}$ is the regression planes normal and s is a steepness scaling parameter.

$$h = \left(\begin{pmatrix} -p_x \\ -p_y \\ 0 \end{pmatrix} \cdot \hat{n} \right) \hat{n}_z \cdot s \tag{3.1}$$

As seen in Figure 3.6, this gives us a surface geometry without triangular holes.

3.4 Surface Cavities

If a tile created from a low field value has opposing neighboring tiles created from higher field values, the overall surface will make a small "dip" at the low tile. This structure will

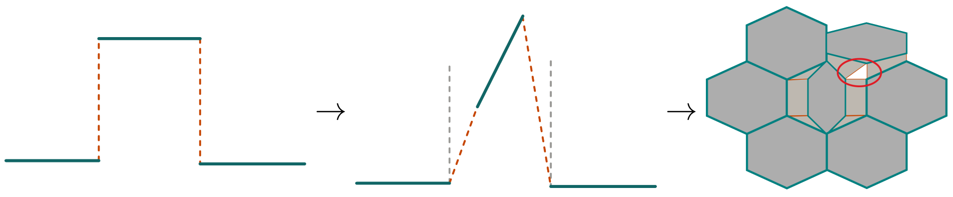


Figure 3.5: Simply rotating the hexagon distorts our rectangles connecting to other hexagons, potentially leaving holes in our geometry which can be seen from the top-down view to the far right.

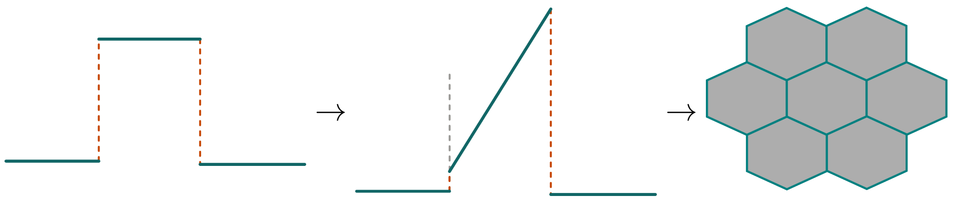


Figure 3.6: Instead of simply rotating the hexagon (Figure 3.5), we translate the vertices of the hexagon along the same axis which keeps the top-down view, to the far right, correct and ensures no holes are created.

henceforth be referred to as a *surface cavity*. If a user tries to sense the surface and the diameter of the surface cavity is less than the diameter of the user's finger, the user will be able to detect the cavity but is likely unable to reach the tile. This issue is visualized in Figure 3.7 with crosses marking spots where a user can not reach. As the user is not able to verify the value of the tile, multiple cavities can be perceived as similar although their actual values might differ drastically.

One possible solution to fix this issue is to increase the diameter of each tile to a reasonably large size such that a user would always be able to reach the surface. Similarly, one would also have to scale the height of the surface such that the steepest surface cavity is reachable with the length of a finger. Each 3D printer has a limit on its printing area, which limits the maximum size of a printed model. Increasing the diameter per tile, therefore limits the maximum resolution of the tile grid one could print, further limiting the number of data points that can be physicalized simultaneously.

Instead, we can utilize the fact that surface cavities are created by low values between high values. If we create another physicalization from the same data but inverted, the previous surface cavities have now become local peaks, which are much easier to reach. Utilizing this, we propose to solve the issue with surface cavities by using the previously

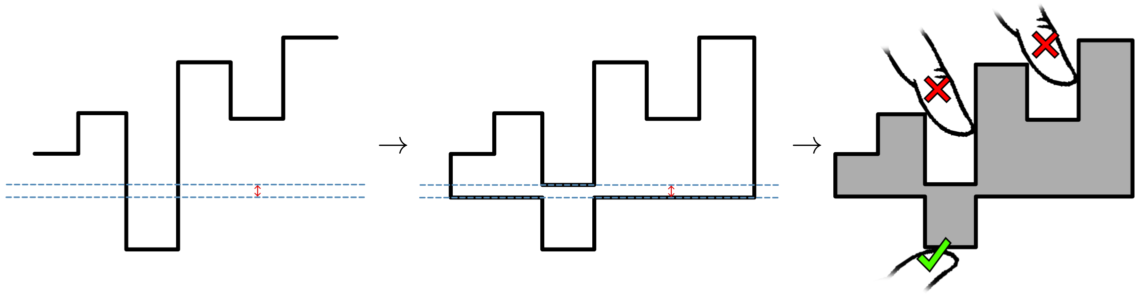


Figure 3.7: Aproach 1: Overcoming surface cavities by making a cut through a *surface cavity* and extruding the geometry towards the cut from both sides. The red distance is a "safety bias" to ensure the printed model has no thin areas that would break. The rightmost image shows that only some cavities will be mirrored while some remain inaccessible.

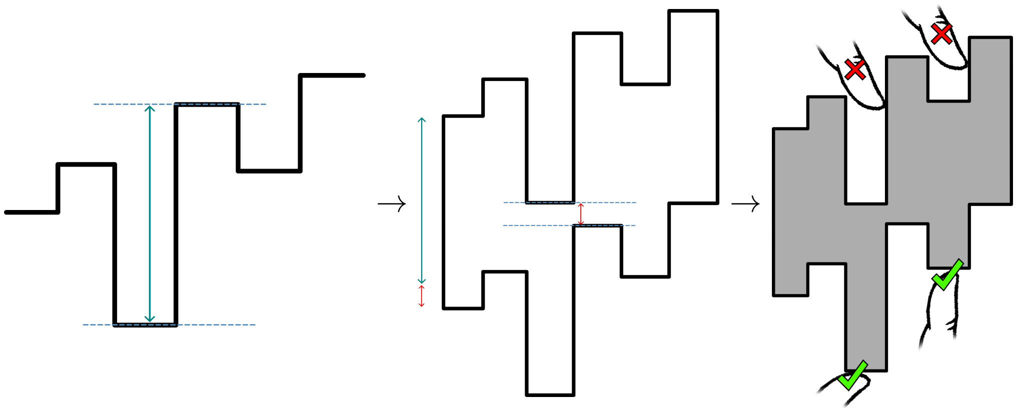


Figure 3.8: Approach 2: Overcoming surface cavities by copying and translating the surface with the smallest amount that does not create a self-intersecting surface (cyan) and a small safety bias (red). The right image demonstrates how using this approach over the previous approach makes all *surface caveties* accessible from the other side of the physicalization.

unused bottom part of our surface to show a physicalization of the inverted data. Whenever the user identifies a surface cavity, they can read the value simply by sensing the same spot on the opposite side of the model.

Initially, the way we proposed to do this was to make a cut through a cavity and extrude the geometry in both directions from the cut, as shown in Figure 3.7. Unfortunately, this not only just solves the issue for one particular surface cavity (illustrated by the checkmark in the figure), but also changes the height of the zero-value tiles on both sides of the cut. By changing the visual representation, the physicalization now gives a different interpretation, as if the underlying data changed. This is a violation of Kindlmann and Scheidegger [120]'s principle of *Representation Invariance*.

Our second approach instead mirrors and translates the top surface to become the bottom surface, as shown in Figure 3.8. In this approach, whenever a user has difficulties reaching a spot (marked using x marks in the figure), the same spot should be reachable from the other side of the model (marked using checkmarks in the figure). While this approach does not change the representation, maintaining the principle of representation invariance [120], this approach runs the risk of having self-intersecting geometry instead. Self-intersecting geometry occurs when a local maximum from the bottom surface passes through a local minimum from the top surface. So to prevent this, we find the largest height difference between two neighboring tiles and separate the top and bottom surfaces with the difference. This ensures that no geometry passes through its mirrored side, but the top and the bottom surfaces can still touch in some corners, like the one at the tile with the biggest difference. So we separate the surface by an additional small safety bias (shown in red) to ensure that the geometry is hollow.

3.5 Orientation

Without mirroring the surface on the bottom of the model, the height of the tiles can easily be understood as the value of the data points. Bigger pillars mean data points with larger values. With the bottom part mirrored, identifying the positive axis is a bit less trivial as every pillar is now always the same size. Fortunately, the top and the bottom tiles are translated in the same direction, so the positive axis can be identified as the common translation axis.

Unfortunately, identifying the horizontal axes and the horizontal orientation of the physicalization is not a trivial task. Unlike a visualization on a screen or paper, there is no "up" or "north" direction for our physicalization. So to help the user identify the coordinate system, we propose encoding a small orientation mark.

If there exists a sizeable area of zero-valued tiles in the physicalization, a small orientation mark depicting the encoded axes can be etched into the flat area. The mark itself can be as simple as a right-angled triangle where the two catheti correspond to the x- and y axes. By sensing the mark, a user can identify whether they are reading the physicalization correctly or adjust the orientation accordingly. A visualization of this mark is given to the left in Figure 3.9.

If no areas large enough to place the mark can be found, a small notch in the corner of the 3D model can also suffice to tell the user the orientation. A visualized of this is given

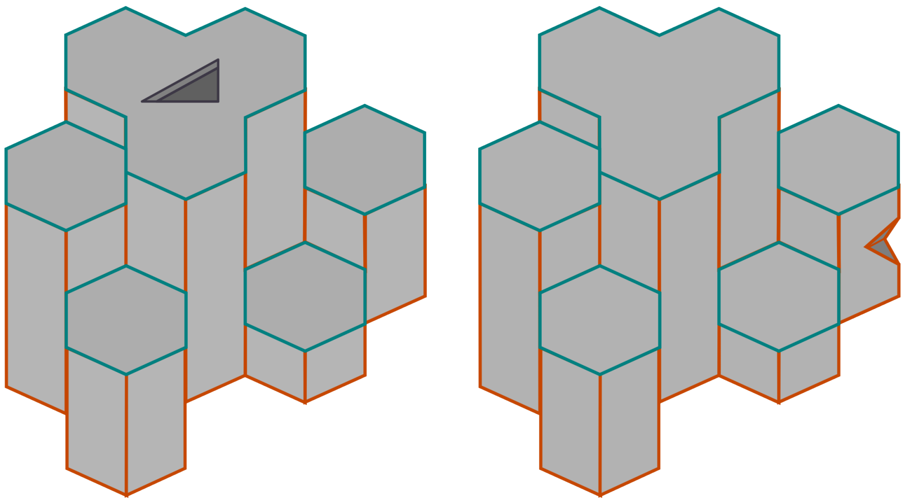


Figure 3.9: To the left: the orientation right-angled triangle indicating the direction of the x- and y axes via its cathethi. To the right: the orientation notch indicating the origin of the coordinate system.

to the right in Figure 3.9. The notch can be understood as the origin of the coordinate system. However, unlike the right-angled triangle glyph, the notch does not encode information about whether to treat the coordinate system as left- or right-handed. It is therefore important to be consistent in choosing coordinate systems or add a secondary glyph describing the coordinate system used, like a letter.

3.6 Summary

To construct the geometry for our physicalization our approach starts with constructing a hexagonal 2D grid from two-dimensional scalar field input data, either directly or by aggregating tabular data into a scalar field similar to a hexplot, further explained in Section 3.1. Each hexagon in our grid, each representing one value in the field, is then translated along a third axis according to its value, further described in Section 3.2. Next, each hexagon connects to its neighbor by adding a rectangle between them. After this, we calculate the convex hull around the zero-valued hexagons, cull the hexagons on the outside of the hull and then extrude the sides of our resulting surface down to form the base of the model.

If the scalar field was created from aggregating tabular points, we can give some context to the underlying points by aligning the top surface of the hexagons with the regression plane of the aggregated points, further described in Section 3.3. In Section 3.4 we highlighted how tall hexagonal pillars surrounding small pillars can lead to difficulties

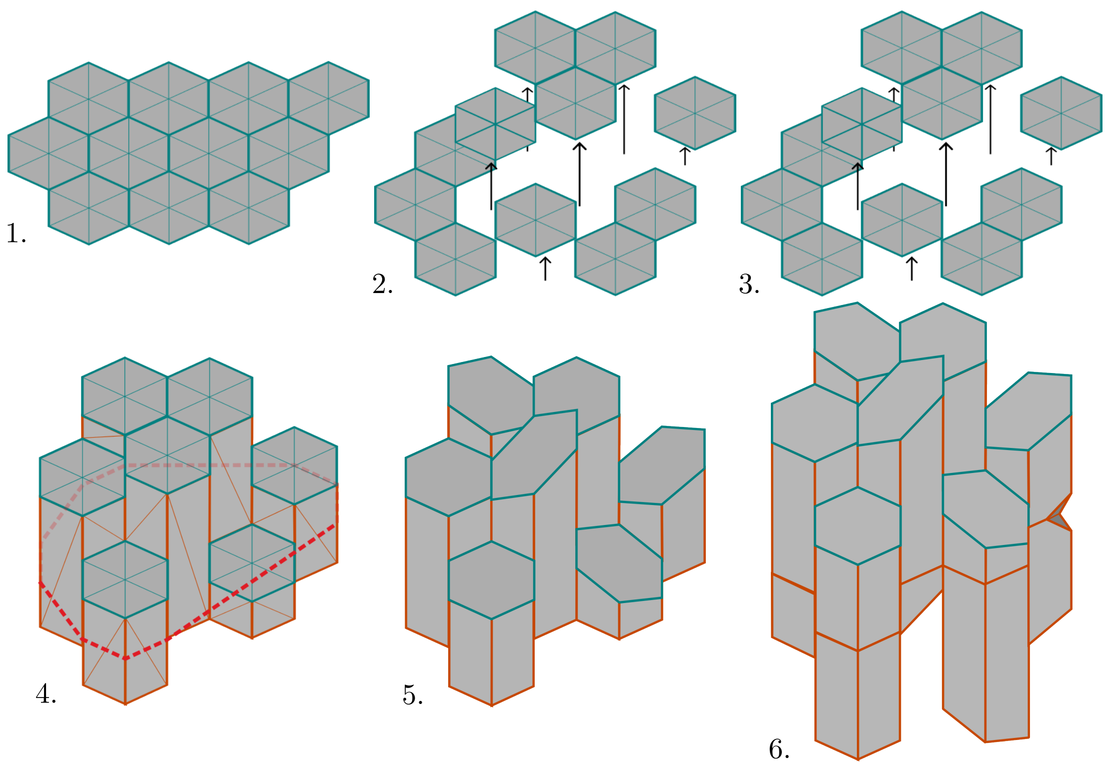


Figure 3.10: An overview of the entire geometry pipeline: (1) A grid of hexagons representing the field is created, (2) The hexagons are translated according to the their corresponding field values, (3) Hexagons are connected with rectangles between them, (4) A convex hull is calculated around the non-zero values and the hull is used to cull hexagons on the outside and extrude down a base of the model, (5) The tiles are aligned with a regression plane of the aggregated points in the underlying hexagonal bin, (6) A mirrored version of the model is attached to its bottom and an orientation notch is indented in its side.

in reading the values. To resolve this issue, we attach a mirrored version of the top surface to the bottom of the model, utilizing how values that are difficult to read due to being obscured by high values are more easily read when the values are flipped.

Finally, in Section 3.5, we improve upon the ambiguity of orienting the model by encoding a notch into the side of the model representing the origin of the coordinate system. The whole pipeline is also summarized in Figure 3.10.

Chapter 4

Haptic Rendering

Our second approach presented in this chapter uses a haptic force feedback device to actively physicalize a two-dimensional scalar field, by mimicking a surface using real-life forces like the normal force, the friction force and the gravity force.

While passive physicalizations have some advantages over classic visualizations, for instance in terms of memorability or information retrieval (see Section 2.2), fabricating a physicalization through AM has some drawbacks, with the biggest one being the time it takes to fabricate the passive physicalization artifact. If the visual encoding is changed or the physicalization should represent some other data, the whole process has to start over. Another drawback could be the geometrical limitations. In our previous approach, we had to encode empty values between non-empty values to ensure that the model remained consistent when interacting with it, and we had to add a vertical displacement to ensure the model was printable.

Active physicalization avoids a lot of these limitations. The physicalization can instantly reflect changes in the dataset or the data representation. The constant feedback loop between a user and the physicalization enables the use of *visual analytics* solutions which are very popular for explorative analysis of data: "Visual analytics combines automated analysis techniques with interactive visualizations for an effective understanding, reasoning and decision making on the basis of very large and complex data sets." [125]

4.1 Encoding Format

Usually, the hardest part about designing an active physicalization revolves around the encoding medium. For this purpose, we decided to encode our physicalization via a force-feedback haptic device. Specifically, we designed our physicalization around the *Falcon* by Novint [103]. Among force-feedback haptic devices, the Falcon is one of the few commercially available at a reasonable price, which is the primary reason we chose this force-feedback device over other ones.

As force-feedback haptic devices can give user feedback in the form of physical forces, more information can be delivered compared to similar wearable haptics techniques discussed in Section 2.4 that usually offer haptic feedback via vibration. However, as explained by Fritz and Barner [117], the tactile sense has a much lower bandwidth compared to the sense of vision, meaning that the amount of information that can be simultaneously processed is lower for tactile displays.

Most current force-feedback haptic devices can only convey one simultaneous positional force, with some solutions also being able to display simultaneous torque or grip forces. In practice, people use multiple touch-points simultaneously to sense surfaces, which is a current limitation of most force-feedback devices. Fortunately, the tactile sense is kinetic, meaning that people can collect information about a surface by moving across it. And while we cannot supply multiple touchpoints with a force-feedback device, we can supply highly accurate force that changes as you move through the physicalization.

Tactile response is also normally two-directional, meaning the way you interact with something will typically change the information you receive. If you press your finger onto a hard and a soft surface, you can typically differentiate between them just by how much resistance the surface gives back. Therefore, we can represent different surfaces by the amount of force we apply to a specific movement using a force-feedback haptic device.

Exploring a virtual environment using an active physicalization also enables us to visualize impossible or non-constructible geometry. For instance, a *Klein bottle* is possible to represent using a haptic simulation but impossible to construct in real life (without intersections). Unlike our previous approach, we do not need to add geometry to ensure the physicalization can be constructed.

In Section 2.1.2 it is mentioned that encoding a 2D scalar field by encoding the value along the third axis makes the reading the value easier but is usually difficult to represent on a computer screen or paper. Plotting software usually does this by creating an isosurface rendered from a viewing angle and using shading to help distinguish the surface, typically via a lightning model like *Phong* or by encoding the surface normal as the color. Unfortunately, projecting the 3D surface to a 2D image often obscures parts of the surface and depth can be hard to distinguish. However, rendering to a haptic device is usually much less involved as a haptic device already offers 3-dimensions of force as the output, thus eliminating the need to rasterize or project to a lower dimensionality. Still, keeping with the idea of representing the scalar field as an isosurface allows us to haptically render it through surface forces instead.

4.2 Normal Force

The main force we need to replicate to create something that will feel like a surface is the normal force. Newton's third law of motion states that if two bodies exact force upon each other, the forces have the same magnitude but opposite directions. If an object is placed upon a table, the force of gravity will push it down into the table; However, the object will remain stationary on the table as an equally great opposing force keeps it stationary. In *classic mechanics*, this force keeping the object on the table is called the *normal force*. The normal force only appears when an object exalts force upon a surface, is perpendicular to a surface, and has the magnitude of the sum of forces from the object in the direction toward the surface. An illustration of how the normal force interacts with objects is shown in Figure 4.1, where F_n is the normal force, F_g is the gravity force and F' is the component of F_g exalted upon the surface. As we are trying to replicate the feeling of a surface in a virtual environment, the normal force will be the most important force we can give the user.

The force of an object is given by $\vec{F} = m\vec{a}$ where m is the mass of the object and $\vec{a}$ is the acceleration currently being applied. If an object is stationary on a surface, its acceleration must necessarily be 0, which means the force $\vec{F} = m0 = 0$ is being applied to the object. In this case, the normal force is negating all the other forces being applied to the object, meaning $\sum_i \vec{F_i} + \vec{N} = 0$, and thus giving the normal as $\vec{N} = -\sum_i \vec{F_i}$.

One additional problem we have not accounted for is the velocity after the impact. If moving towards a surface with a constant speed, the normal force would be 0, meaning only applying the normal force would not change the velocity. Newton's first law of motion states that a body that is stationary or is moving with a constant speed does not

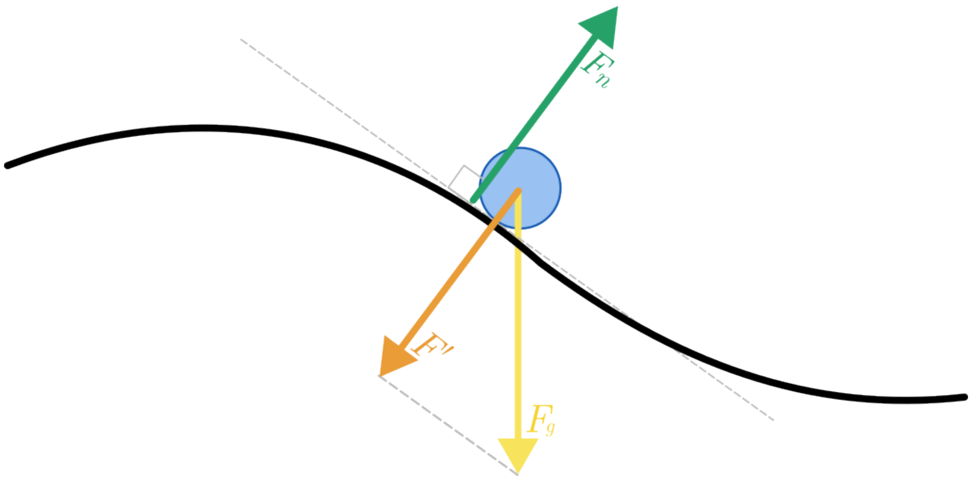


Figure 4.1: In this example, the force of gravity F_g works upon the sphere, which makes the sphere exalt force upon the surface in the direction of the surface, labeled F'. Newton's third law then states that the surface exalts an opposing force back upon the sphere with the same magnitude but in the opposite direction. F' is perpendicular to the surface tangent at the collision point, which gives a normal force of the same magnitude but in the opposite direction $\vec{F}_n = -\vec{F}'$.

have a force acting upon it. This means, at the point of impact, a normal force would be applied to the object that would give a negative acceleration canceling out the velocity. If the impact happens over the time of Δ seconds, the normal force acting upon the object in those Δ seconds would then be given as

$$\vec{N} = -\sum_i \vec{F}_i - \frac{\vec{v}}{\Delta m} \tag{4.1}$$

where $\vec{v}$ is the velocity of the object and m is the mass.

Due to this, finding the normal force is trivial if the haptic device can detect the force applied by the user. However, most haptic devices, like the *3DOF* one we are working with, are limited to positional and rotational sensors. Acceleration can be estimated using position queries, but calculating the force also requires us to know the weight of the haptic device's end-effector (end-effector here refers to the physical part the user interacts with on a haptic force-feedback device). Additionally, we would have to account for outside factors like air resistance or friction in the physical hardware of the device. Fortunately for us, we only have to apply the normal force while the user is pushing against the surface, which we can approximate as when the user is inside of the surface, as moving from a position outside the surface towards a position inside the surface must have involved applying some force to the device.

Now we need to find the magnitude of the force that should be applied. Initially, we can define the force by a user-controlled parameter which we will call s_f. If the user-controlled parameter is much greater than the magnitude of the normal force required to stop the velocity, the result is that the user will be pushed back away from the surface instead of stopping on the surface; the higher the parameter the faster the speed at which the user gets pushed back. A quick workaround to this issue is to simply scale the normal force with the distance from the surface.

Looking at Equation 4.1 we can see that the elapsed time of the impact directly affects how much normal force is applied. If we are already scaling the normal force with the distance into the surface, simply extending the period where we apply the normal would then increase the time of impact. Softer surfaces can therefore be simulated by increasing the distance of which we apply a fraction of the normal.

4.3 Surface Height and Normal Replication

Whenever the haptic device's *probe*, meaning the virtual representation of the haptic device's end-effector, finds itself beneath the surface in the virtual environment, we want to apply a normal force back to the end-effector, pushing the end-effector in the opposite direction and out of the surface. We want our surface to vary with the scalar field and, therefore, define our surface as a plane where the scalar field determines the relative height offset above the plane.

$$uv(p) = \begin{pmatrix} \dfrac{p \cdot \hat{t}}{2w_l} \\ \dfrac{p \cdot \hat{b}}{2h_l} \end{pmatrix} \tag{4.2}$$

We need to determine the relative position of the probe to the surface. To do this, we first project the position down onto the plane of the surface, shown in Equation 4.2 where p is the position, $\hat{t}$ and $\hat{b}$ are the plane's tangent and bitangent unit vectors, and w_l and h_l are the width and height of our simulation bounds. Projecting the position gives us the position in the plane's basis, which we can treat as texture coordinates within the plane. Then, doing a texture lookup using the texture coordinates on the scalar field gives us the bilinearly interpolated scalar field value, which we use as an offset from the

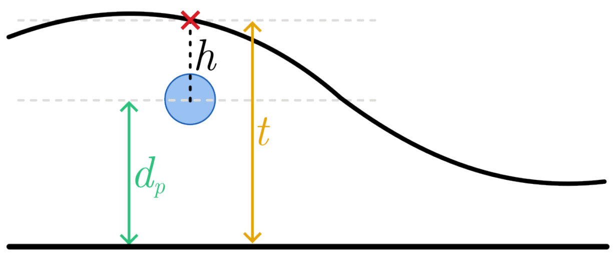


Figure 4.2: Our surface in our virtual environment is defined as a plane, offset by a heightmap. To determine the probe's relative height to the surface, we first find the distance from the probe to the plane d_p, which according to Equation 4.3 can be found as $d_p = (p - p_l) \cdot \hat{n}_l$ where p is the probe position, p_l is a point the plane passes through and $\hat{n}_l$ is the plane normal. After finding the distance to the plane, we subtract with the sampled value t at the probe's texture coordinates, found using Equation 4.2. The final relative distance h is then given as $h = d_p - t$.

probe's distance to the plane. The final relative height of the probe above the surface is then calculated in Equation 4.3 where p is the probe's position, p_l is a point passing through the plane, $\hat{n}_l$ is the plane's normal, t is the texture sampling function and s is a user-defined surface scaling parameter. An illustration is given in Figure 4.2 where d_p is the distance to the plane $(p - p_l) \cdot \hat{n}_l$, t is the sampled scalar field value $t(uv(p))$ and h is the resulting height above the surface.

$$h(p) = (p - p_l) \cdot \hat{n}_l - t(uv(p)) \cdot s \tag{4.3}$$

When the relative height above the surface is negative ($h < 0$), it means the probe is situated beneath the surface. This is when we want to apply a normal force to the haptic device. As the normal force is collinear with the surface, we set the normal force from the surface's normal multiplied by the previously mentioned user-controlled parameter s_f. To calculate the surface's normal we first have to find the surface's gradient. The gradient is calculated using the partial derivatives of the height sampling function, which again is approximated using finite differences, demonstrated in Equation 4.4 where p is the probe's position.

$$\nabla(p) = \begin{pmatrix} \dfrac{\partial h}{\partial x}(p) \\ \dfrac{\partial h}{\partial y}(p) \end{pmatrix} \tag{4.4}$$

As the field is a 2D scalar field, its gradient is also limited to two dimensions. The surface normal vector is therefore created from the cross product of two unit vectors lying in the plane formed by the partial derivatives, as shown in Equation 4.5 with ∇_x and ∇_y being the gradient in the x and y direction and p being the position.

$$\hat{n}(p) = \left\| \begin{matrix} 1 \\ 0 \\ \nabla_x(p) \end{matrix} \right\| \times \left\| \begin{matrix} 0 \\ 1 \\ \nabla_y(p) \end{matrix} \right\| \tag{4.5}$$

Finally, the normal force from the surface, illustrated in Figure 4.6 as $\vec{F}_n$, is calculated as

$$\vec{F}_n(p) = \hat{n}(p) \cdot s_f \tag{4.6}$$

where s_f is the user-controlled force parameter.

4.4 Friction and Gravity

Fritz and Barner [117] emphasize that friction is an important part of what makes a real surface feel like a surface. The virtual friction force mimics what in classic mechanics is known as *dry friction*. Dry friction is a force that opposes the lateral movement of two surfaces. Dry friction is colinear to the surface, pointing in the opposite direction of the velocity with a magnitude dependent on the velocity magnitude and is typically a fraction of the normal force magnitude. An illustration is given in Figure 4.3. Dry friction is further divided into *static* and *kinetic* friction. With increased lateral movement between the surfaces, the friction between them transitions from static to kinetic friction, with the magnitude of static friction being higher than that of kinetic friction.

Our friction force simulation is based on the approach described by Fritz and Barner [117] in their paper, with only some minor adjustments. In the approach proposed by Fritz and Barner [117], when the probe first enters the surface, a *sticktion point p_s* is placed on the surface position of the probe. In the next simulation steps, if the current probe's position is within a radius of the sticktion point, with the radius being equal to a constant set by the implementation, they apply static friction. Static friction is calculated as a vector from the probe position to the sticktion point multiplied by the magnitude of the normal force $|\vec{F}_n|$ and a static friction coefficient μ_s also set by the implementation. The static friction parameter μ_s can be anything that fulfills $0 \leq \mu_s \leq 1$. This means static friction always pulls the probe towards the point it first entered the surface, which is visualized in Figure 4.4. If instead, the probe position is outside the radius of the sticktion point, they apply kinetic friction which is calculated similarly but multiplied by

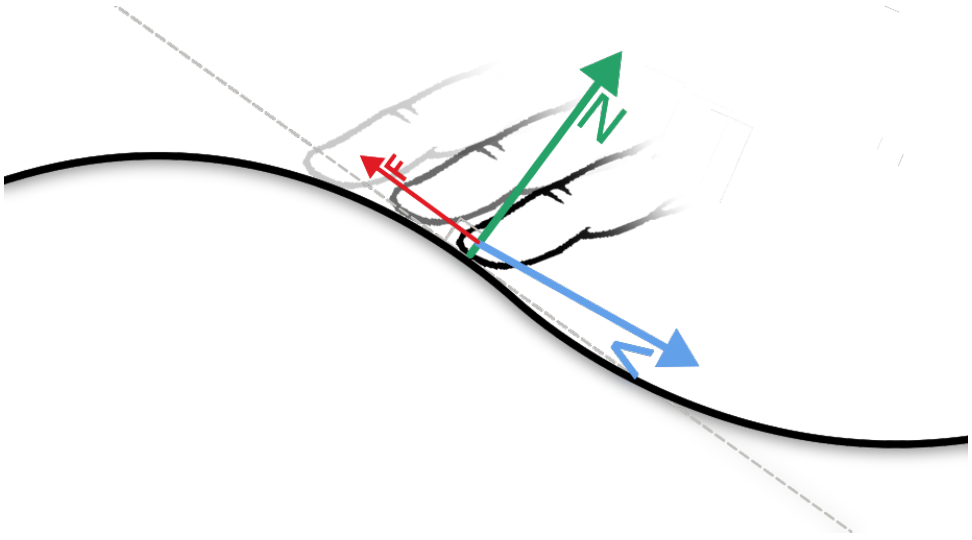


Figure 4.3: Dry friction (F) is a force orthogonal to the surface normal force (N), with a fractional magnitude of the velocity (V) and aligned in the opposite direction of velocity.

a kinetic friction coefficient μ_k instead of the static one. As kinetic friction is lower than static friction, μ_k has to satisfy $\mu_k < \mu_s$.

Our approach is mostly similar, but instead of using the distance to the sticktion point, we opted to use the magnitude of the velocity $\vec{v}$ instead, as real-life dry friction is only dependent on velocity and not position. Velocity can be calculated as the change in position since the last simulation step, normalized over time, illustrated by the blue vector in Figure 4.5. The proposed radius then becomes a threshold in our approach instead, which we set equal to $|\vec{F}_n| \cdot \mu_s$ to simplify the equation. If we never move the sticktion point our kinetic friction will scale with the distance to the point where we first intersected with the surface, but real friction is not dependent on how long you have been moving across a surface. If we add the requirement that μ_s should be non-zero $(0 < \mu_s \leq 1)$, we can utilize the fact that when we apply kinetic friction the velocity necessarily also have to be non-zero, as kinetic friction is only applied when $|\vec{F}_n| \cdot \mu_s < |\vec{v}|, \quad 0 < \mu_s \leq 1$. We can then find the direction of our velocity $\|\vec{v}\|$, which gives us the direction we are moving. And we can "drag" the sticktion point behind us by setting it to our current probe's position displaced by the negative direction of the velocity, which is visualized to the right in Figure 4.5. The new position of the sticktion point is calculated as $p_o - \|\vec{v}\| \cdot |\vec{F}_n| \cdot \mu_s$ where p_o is the probe's current position projected to the surface. Finally, the dry friction force, illustrated as $\vec{F}_{fr}$ in Figure 4.6, can be calculated using Equation 4.7:

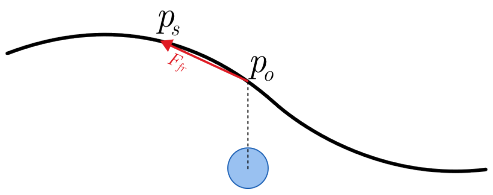


Figure 4.4: For static friction, the friction $\vec{F}_{fr}$ is collinear with the vector from the current probe's (shown as a blue circle) surface-projected position p_o, towards the sticktion point p_s.

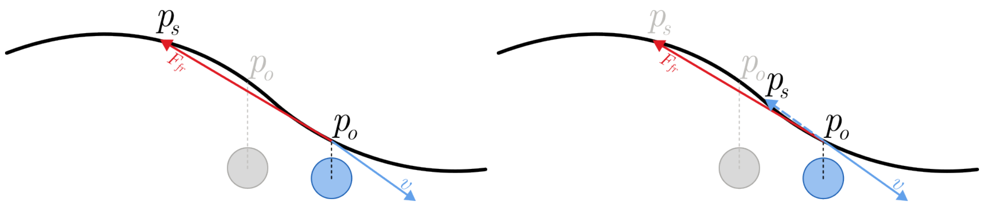


Figure 4.5: Kinetic friction is calculated similar to static friction, as shown in Figure 4.4, but afterwards "drags" the sticktion point p_s behind the surface position p_o using an offset in the direction of the negative velocity. Marked in gray are the positions from the last simulation step.

$$\vec{F}_{fr} = \begin{cases} (p_s - p_o) \cdot |\vec{F}_n| \cdot \mu_k, & |\vec{F}_n| \cdot \mu_s < |\vec{v}| \\ (p_s - p_o) \cdot |\vec{F}_n| \cdot \mu_s, & \text{otherwise} \end{cases} \quad , \quad 0 \leq \mu_k < \mu_s \leq 1 \tag{4.7}$$

Depending on the haptic device and the API used in the implementation, the end-effector may or may not apply a stationary force keeping the end-effector in place by default, which is the case for the Novint Falcon we are using. If the end-effector is stationary when no forces are applied, adding a constant artificial gravity force could help with the illusion of a real surface. Simulating a gravity force can be done simply by adding a constant force on the down-axis of the haptic device at every step of the simulation, for instance as

$$\vec{F}_g = \begin{pmatrix} 0 \\ 0 \\ -9.81 \end{pmatrix} \tag{4.8}$$

, which is illustrated as $\vec{F}_g$ in Figure 4.6.

4.5 Local Encoding of High-Level Properties

Because our haptic device has a singular position input and a singular force output for each frame, our probe is essentially a single point in the virtual environment. People

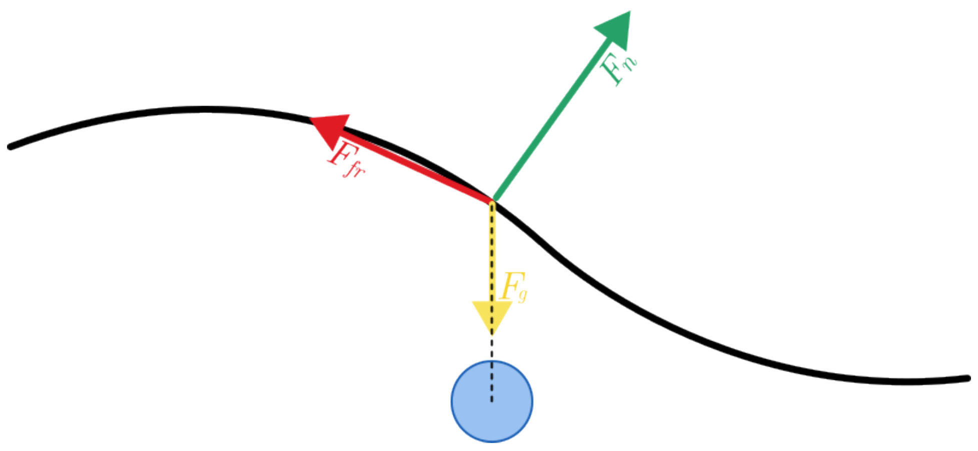


Figure 4.6: This figure shows an overview of the forces we apply where $\vec{F}_n$ is the normal force, $\vec{F}_{fr}$ is the friction force and $\vec{F}_g$ is the gravity force. The blue sphere indicates the probe and the strippled line is its surface projected position.

sense physical objects by moving multiple local sensing points across a surface, collecting simultaneous local information about the surface and using this to build a global image of the object. As our probe is a singular point, we are more limited in the amount of simultaneous information about the surface we can supply. However, we can still make use of how people dynamically collect information about a surface by moving across it, which we have done by encoding global information locally as a 3D volume situated above the surface. The idea which is visualized in Figure 4.7 is that as the user moves through the volume, from the top of the volume towards the surface, the user should detect a surface that interpolates between a smoothed overview surface and a higher detailed one. This gives the user an understanding of the overall structure of the surface by moving through the volume in a higher region, and more local and detailed values by moving through a lower region. We placed the most smoothed representation in the top region of the volume and the most detailed representation in the bottom region, which means the bottom region could be as detailed as our previous surface representation. We then place our previous surface representation directly beneath the volume such that a user can smoothly move from a course overview toward the actual surface with gradually increasing accuracy.

To smoothly transition from the volume to the surface, we keep the notion of a surface normal in the volume as well. In Section 4.2 we highlighted how simply scaling the normal force can make the surface feel softer, thus by setting the normal in the volume to a fraction of the normal on the surface the user should still be able to differentiate them as different objects.

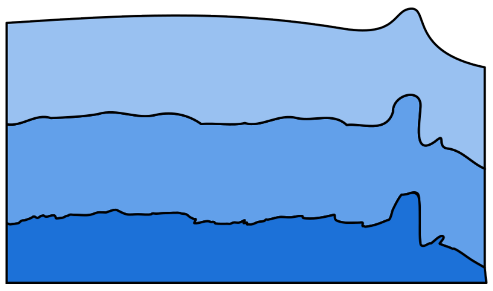


Figure 4.7: The volume consist of progressively smoothed surfaces stacked on top of each other

By taking progressively smoothed levels of the surface and stacking them on top of each other, we can treat the surface stack as a volume. These smoothed levels can for instance be iteratively acquired from the previous level by applying a Gaussian smoothing filter or, using scale-space theory, can be defined as the scale-space representation $L(x, y, t) = g(x, y, t) * f(x, y)$ with increasing scale parameters t $(t = 0, t = 4, t = 16, ...)$ where $L(x, y, t)$ is the convolution of the image representation $f(x, y)$ and the Gaussian kernel $g(x, y, t) = \frac{1}{2\pi t} e^{-(x^2+y^2)/2t}$. We then define the density for the volume as the linearly interpolated height between the different levels, shown in Equation 4.9 as $d_i(p)$ with $h_i(p)$ being the height-sampling function for the level i and t is the interpolation parameter between the levels.

$$d_i(p) = (1 - t) \cdot h_i(p) + t \cdot h_{i+1}(p) \tag{4.9}$$

The interpolation parameter t can be calculated by taking the difference in the height between the underlying planes for the height-sampling function, shown in Equation 4.10 where $h'(p)$ is a simplified version of Equation 4.3 taking only the underlying plane into account and discarding the sampled height.

$$t(p) = \frac{h'_0(p) - h'_i(p)}{h'_{i+1}(p) - h'_i(p)}, \quad h'(p) = (p - p_l) \cdot \hat{n}_l \tag{4.10}$$

We then use $d_i(p)$ to calculate the gradient $\nabla(p)$ in the volume similarly to how it was done in Section 4.3, only this time using the finite differences of the interpolated height instead of the height directly, as shown in Equation 4.11.

$$\nabla(p) = \begin{pmatrix} \frac{\partial d_i}{\partial x}(p) \\ \frac{\partial d_i}{\partial y}(p) \end{pmatrix} \tag{4.11}$$

Treating the normalized gradient as the volume normal then gives us a normal vector at any point within the volume. As the volume density is sampled from a 2D scalar field, the gradient calculated from it does not change over the third up-axis. To account for this, we simply replace the up-axis in the normal with the height from the bottom surface within the volume. This gives us the final equation for our volume normal force, shown in Equation 4.12, where s_f is the user-controlled force parameter and k is a scaling parameter set on the implementation side to an amount sufficiently smaller than 1 such that the surface and the volume can be differentiated (e.g. $k = \frac{1}{2}$).

$$\vec{F}_v(p) = \left\| \begin{matrix} \nabla_x(p) \\ \nabla_y(p) \\ h_0(p) \end{matrix} \right\| \cdot s_f \cdot k, \quad 0 \leq k \leq 1 \tag{4.12}$$

4.6 Summary

To create a surface representation of the two-dimensional scalar field that can be rendered on a haptic force-feedback device, we rely on replicating three forces from classical mechanics: the normal force, the friction force and optionally the gravity force, which is illustrated in Figure 4.6. The normal force further explained in Section 4.2 is the most important force to replicate, as this is the force that prevents objects from passing through one another. When the user moves the probe in the virtual environment beneath the surface made from elevating a plane using the scalar field as a heightmap, we apply a normal force to push the probe back out. As detailed in Section 4.3, our surface normal force is calculated by taking the cross product of two unit vectors lying in the tangent plane, where the tangent plane is defined by the gradient calculated using central differences.

Next, detailed in Section 4.4, we simulate dry friction over the surface. We do this by creating a *sticktion point* at the location the probe first entered the surface, which we gravitate the probe towards. To simulate kinetic friction, which is dry friction in motion, we move the sticktion point behind the probe as it traverses the surface such that the friction amount is independent of the distance traveled across the surface. Our third and final force, the force of gravity, is afterward optionally applied as a constant down-pulling force.

The haptic device can convey one force at a time, which makes our haptic rendering give a local view of the surface. In Section 4.5, we improve upon this by encoding global information about the surface on a local level using a volume built from layering increasingly smoothed versions of the surface on top of each other. Moving through the smoother part of the volume reflects large global structures and clusterings in the scalar field while moving through the more detailed part reflects the smaller and more local details.

Chapter 5

Implementation

This chapter covers our implementation of the methods specified in Chapters 3 and 4. Both the physicalization and the haptic rendering parts are implemented as an extension of the project used for the implementation of HoneyComb plots by Trautner et al. [23]. The source code for the project can be found at *github.com/andesyv/tangible-scalar-fields*.

5.1 Physicalization

Most of the implementation for the physicalization was done using C++ and OpenGL in a *General-Purpose computing on Graphics Processing Units (GPGPU)* approach, meaning parts of the tasks are offloaded to the GPU.

The STL format stores surface geometry in triangle primitives using triangle vertices and vertex normals (see Section 2.3.2). This is the same way OpenGL handles geometry, meaning if it can be rendered in OpenGL, it can be stored in an STL file.

5.1.1 Geometry

Following how it was described in Chapter 3, our pipeline starts with a scalar field which is created by aggregating tabular data in a hexagonal grid similar to a normal hexplot implementation. We then generate a grid of hexagons which we translate according to each hexagon's corresponding scalar field value and connect all the hexagons together with

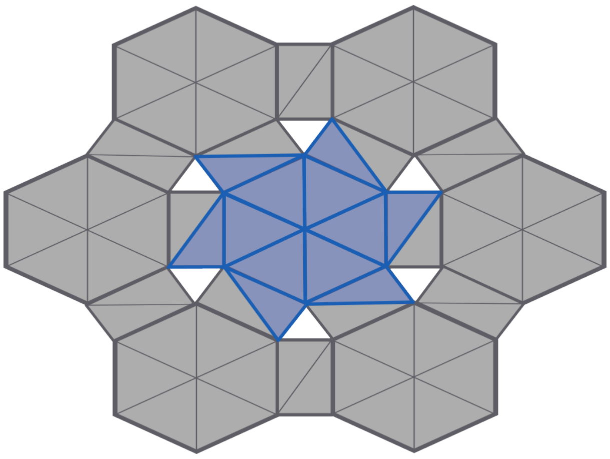


Figure 5.1: The blue shaded triangles represent the work group for the middle hexagon.

rectangles made from two triangles. The geometry generation process is highly individual per hexagon, so we perform this main grid generation on the GPU via OpenGL *Compute Shaders*. A compute shader is a custom shader program running entirely on the GPU that is specialized to perform general-purpose computing instead of rendering. A compute shader is invoked with a set amount of work groups in which the tasks are divided among work groups and run in parallel. Each work group is further divided into a set amount of local invocations which are also run in parallel.

As the number of work groups to invoke is given as a three-dimensional integer vector in the OpenGL implementation, we set each work group to correspond to one hexagon in the grid and set the invocation space to $\lceil n^{\frac{1}{3}} \rceil^3$ where n is the count of hexagons in the grid, to ensure that at least one work group is invoked per hexagon. Between two hexagons, only one rectangle needs to be placed, meaning each workgroup can create half of the triangles in the rectangle connecting to the neighboring hexagon. As each hexagon consists of six edges, this gives a total of twelve triangles and twelve local invocations, with half of them creating the geometry for the inside of the hexagons and half of them creating the geometry connecting to the neighboring hexagons. The layout of the triangles within a workgroup is displayed in Figure 5.1.

After the main geometry is generated, we calculate the convex hull around the hexagons with non-zero scalar values, which are afterward used to cull the hexagons outside of the convex hull. The convex hull is calculated using *Graham scan* [122] on the CPU, while hexagon culling happens on the GPU. After calculating the convex hull, a compute shader is run that simply discards a hexagon if it is outside the hull.

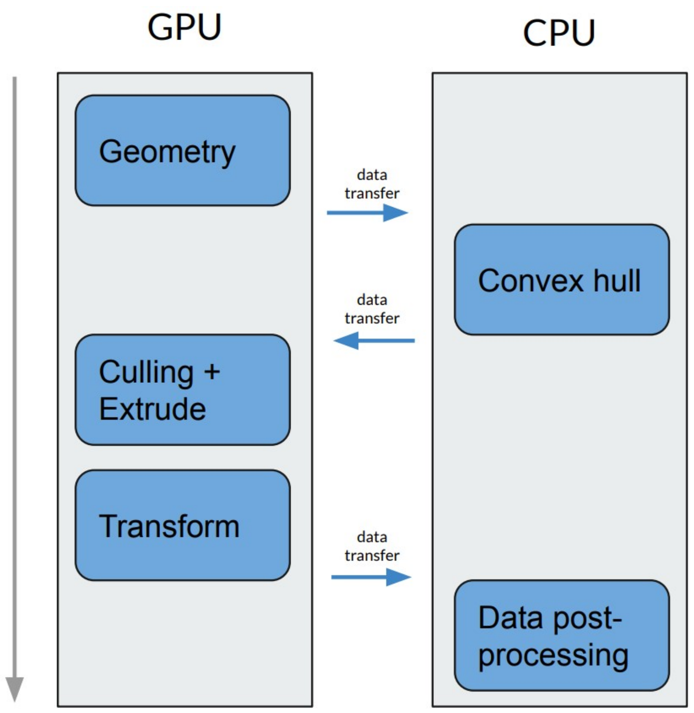


Figure 5.2: An overview of the whole physicalization workflow and how work is divided between the GPU and CPU.

After culling outside hexagons, vertices along the convex hull are extruded down to form the base of the model using a compute shader. At this point, we are left with a single hole in the shape of a polygon. If we are not encoding a mirrored bottom surface (see Section 3.4), this is the point where we fill the hole geometry ourselves or leave it to the slicer. Guo et al. [123] notes that flat polygonal holes can often be filled using simple triangulation algorithms, which we can utilize by extruding our base down to the same plane. Since our hull extrusion happens on the GPU, it makes sense to leverage the GPU for the triangulation as well, such that no CPU-GPU transfers are required. For instance, using the triangulation approach by Qi et al. [126]. However, most current slicer software can also fill the hole automatically.

The mirrored bottom surface (Section 3.4) is constructed similarly to the top surface, except that the geometry is flipped. Therefore, we construct the bottom surface geometry in the same pass as the top surface by invoking twice the amount of compute shaders and flipping the triangle winding order for the bottom surface. Extrusion of the boundaries also happens for each surface, but the bottom boundary is instead extruded up. After constructing both surfaces and extruding their boundaries, both surfaces overlap as they were constructed in the same coordinate space. But, they also share a similarly shaped hole, only flipped. Therefore, in an additional last step, we connect the top and the bottom surfaces by translating both surfaces away from each other until the boundary of their holes overlaps.

5.1.2 Orientation Notch

Section 3.5 mentions two approaches to encode orientation. The first approach places a right-angled triangle on a sufficiently large flat surface to indicate the direction of two axes. The second approach simply makes a notch in the side of the model to indicate the origin of the reference system.

We ended up opting for the second approach because the way we handle the mirrored bottom surface makes it easier to implement. After the top and the bottom surfaces end up with overlapping holes, both surfaces end up sharing a common set of vertices aligned on a plane through the middle of the model. Simply moving the boundary vertices on the top and bottom surface introduces holes in the side of the model. By filling the created holes with geometry from an inverted shape, we can encode shapes on the side of the model in a fashion similar to a *difference operation* in CSG. In our case, this would be the difference operation between our model and a ”wedge”-like model. A ”wedge”-like model can be represented with two intersecting planes, and since our surfaces are mirrored, we can construct the ”wedge”-like hole from both sides by taking the difference operation of our surface and a plane.

5.2 Haptic Rendering

Our implementation was made for the *Novint Falcon* [103], but due to their similarities should work with any 3DOF force-feedback haptic device.

There exist some software available for interacting with haptic force feedback devices. Among the most popular ones are the *OpenHaptics* toolkit by 3D Systems [127] and the *CHAI3D* open-source framework by Conti et al. [128]. *OpenHaptics*, despite its name, is a closed source software made by the same developers of the *Phantom* devices, designed to work with their devices. *CHAI3D* is instead a lightweight open-source framework with support for more haptic devices, including the *Phantom* devices. *CHAI3D*’s support for the *Phantom* based devices comes from the *Haptic SDK* by Force Dimension [129]. *CHAI3D* includes some higher-level abstractions for common usages of a haptic device, but since we are interested in implementing the whole pipeline ourselves we opted for using the lower-level SDK by Force Dimension [129] instead.

To get a seamless experience using a haptic force feedback device, instructions need to be sent to the device at a high rate, typically around 1000 instructions per second due to the tactile sense having a high temporal resolution [13, ch. 5.7]. To prevent other tasks in the application from slowing down the interaction rate, we dedicate a single thread to interact with the haptic device. At each simulation step, the thread: starts by fetching the current position of the haptic input device's probe, converts the position from the device's space to local space, calculates the force that should be applied to the haptic device at its current position, and finally converts the force back to its local space and applies it to the device.

When the relative sampled height from the surface is positive, according to the approach in Section 4.3, the position is in the *air*, and no forces other than gravity should affect the probe. Otherwise, when the height is negative and the probe is beneath the surface, the surface normal is sampled from the surface using the normalized gradient, which is approximated using central differences. The normal force from the surface is then set as the surface normal multiplied by the force amount specified via a user parameter. To prevent big jumps in the force felt on the user end-effector caused by entering and exiting the surface, we multiply the force with an additional scalar that smoothly interpolates from 0 to 1 over a small distance into the surface.

The sampled height and the resulting approximated gradient are calculated from the surface data, which is stored as discretized height values in a 2D grid. However, height sampling needs to be continuous, and the height thus gets bilinearly interpolated between neighboring height values within the discretized field. This is shown in Equation 5.1 as $h_n(u, v)$ with u and v being the texture coordinates and T_n being the texture samples for a level of detail n.

$$h_n(u, v) = \begin{pmatrix} (1-u)(1-v) \\ u(1-v) \\ (1-u)v \\ uv \end{pmatrix} \begin{pmatrix} T_n(u, v) \\ T_n(u+1, v) \\ T_n(u, v+1) \\ T_n(v+1, u+1) \end{pmatrix} \tag{5.1}$$

As mentioned in Section 4.5 we construct our volume by stacking progressively smoothed levels of the surface. Since the surface can be stored as a heightmap, one could construct these levels by iteratively applying a smoothing filter to the texture, e.g.

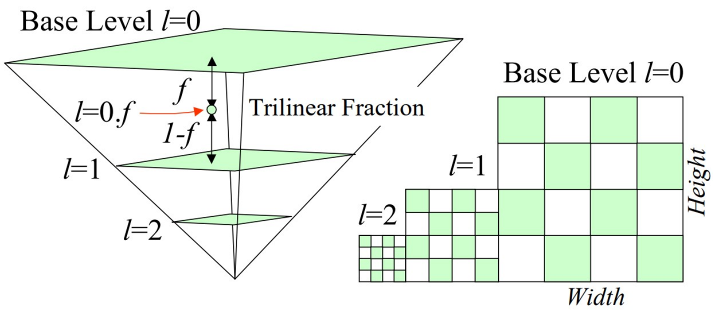


Figure 5.3: The figure shows how the different texture sizes are represented in a pyramid where the sampled texture is trilinearly interpolated between the two closest textures [130].

by convolution with a Gaussian kernel. However, as convolutions are computationally expensive, we instead opted for using the less accurate but more effective method of using mipmaps. Mipmaps, visualized in Figure 5.3, are sets of images as levels where each level represents a lower resolution version from the previous level. Each pixel in the higher level mipmap represents the average of 4 pixels in the previous mipmap, as shown in Equation 5.2. Since our height gets bilinearly interpolated between the pixels, the result of sampling the height from a higher level mipmap gives the same result as applying an iterative linear smoothing kernel.

$$T_n(i,j) = \frac{1}{4}\begin{pmatrix} T_{n-1}(2i,2j)+ \\ T_{n-1}(2i+1,2j)+ \\ T_{n-1}(2i,2j+1)+ \\ T_{n-1}(2i+1,2j+1) \end{pmatrix} \tag{5.2}$$

Chapter 6

Results

6.1 Physicalization Artifacts

To test the physicalization pipeline we decided to physicalize airports across Europe. The physicalization was printed on a Prusa i3 MK3S+ 3D Printer using PLA filament. The dataset we used was The Global Airport Dataset [131], which is a free dataset of tabular data consisting of longitude and latitude coordinates of airports across the globe. We further scoped the data down by limiting it to countries within Europe and removed outliers such that the shape of the remaining points should be easily recognizable as Europe. We then converted the points to a scalar field and a corresponding hexplot, which is displayed in Figure 6.1.

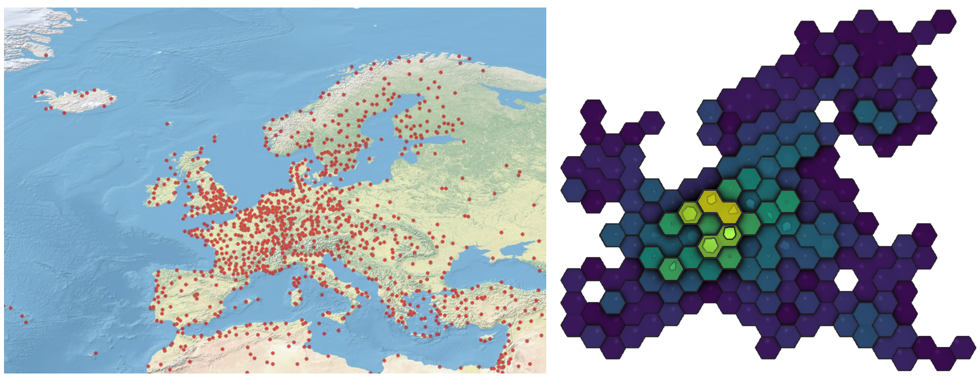


Figure 6.1: The input data of airports across europe [131] and the corresponding hexplot generated from the implementation by Trautner et al. [23]. In the hexplot, longtitude and latitude are used as x and y coordinates.

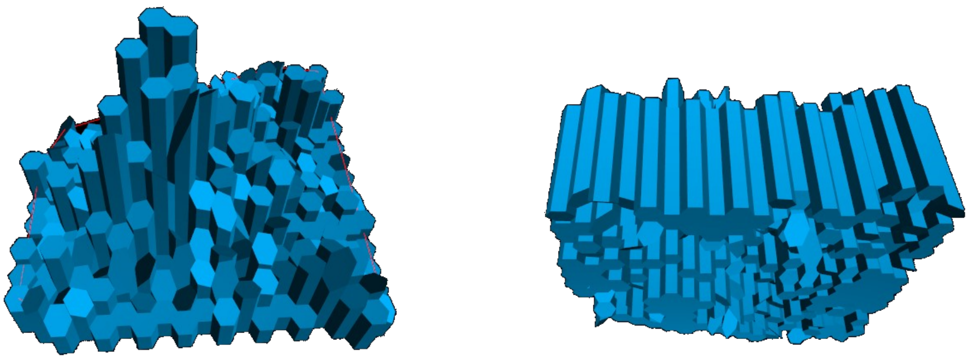
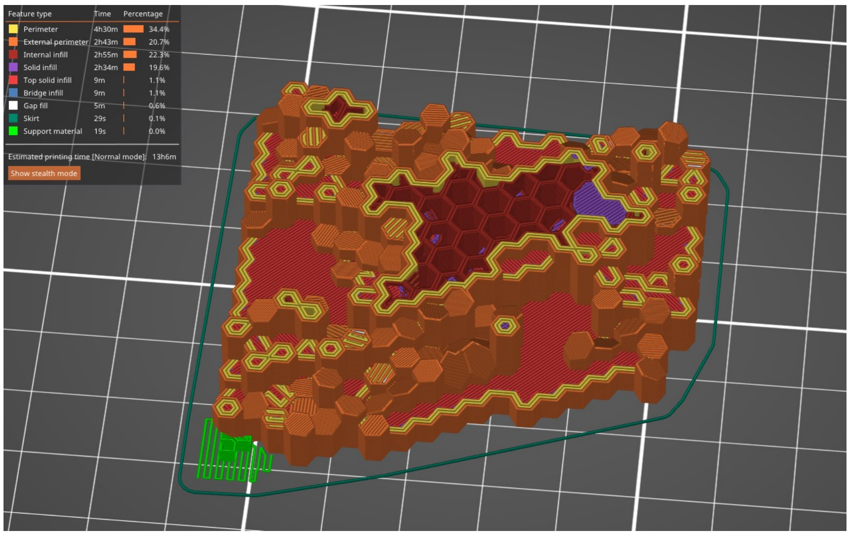


Figure 6.2: The generated geometry from our approach. The model on the left has a flat bottom surface and the one on the right has a mirrored bottom surface.



	Feature type	Time	Percentage
🟨	Perimeter	4h30m	34.4%
🟧	External perimeter	2h43m	20.7%
🟥	Internal infill	2h55m	22.3%
🟪	Solid infill	2h34m	19.6%
🟥	Top solid infill	9m	1.1%
🟦	Bridge infill	9m	1.1%
⬜	Gap fill	5m	0.6%
🟩	Skirt	29s	0.1%
🟩	Support material	19s	0.0%

Estimated printing time [Normal mode]: 13h6m

Figure 6.3: The generated machine tool path created with *PrusaSlicer* [98]

Using the same scalar field and grid layout as in Figure 6.1, we then used our technique to generate 3D models and exported them as STL files which were then printed. Figure 6.2 shows the 3D models of the two different models we generated. The one to the left is one created with a flat surface, and the one to the right is the same as the one to the left but with the bottom surface mirrored using the technique described in Section 3.4. Next, the STL file was imported into the *PrusaSlicer* [98] slicer software and a toolpath for the printer was generated, which is displayed in Figure 6.3. Finally, the toolpath was exported and printed on a Prusa i3 MK3S+ 3D Printer using PLA filament in different colors.

Two of the final printed artifacts with flat bases can be seen in Figure 6.4, which were

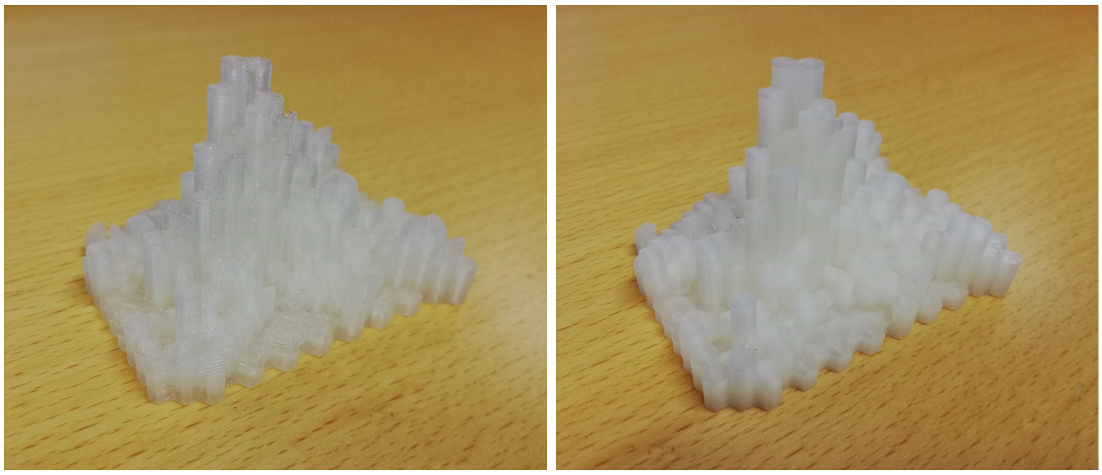


Figure 6.4: The printed result of the model with a flat bottom surface, printed in two separate types of transparent PLA filament

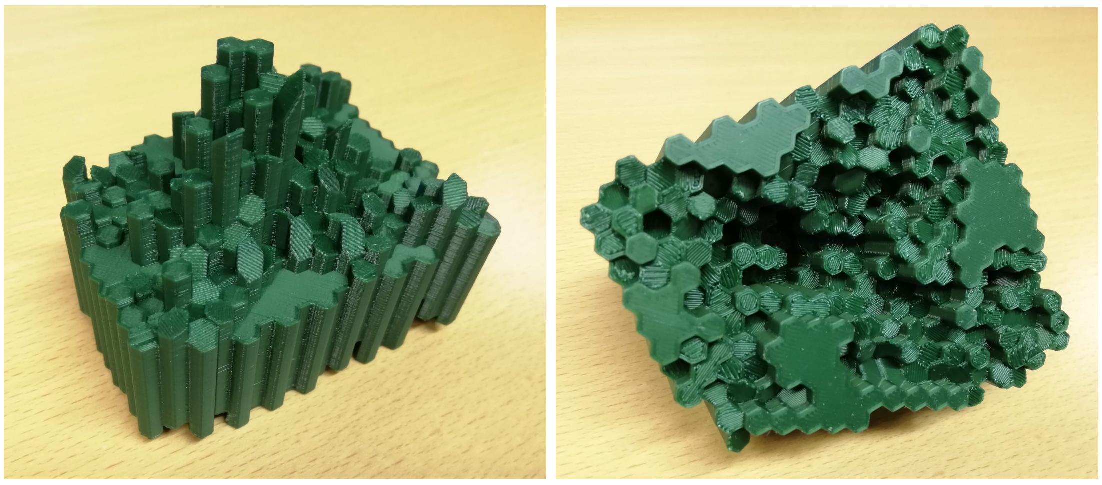


Figure 6.5: The printed result of the model with a mirrored bottom surface

both printed with clear PLA filament but from different manufacturers. The artifact to the right also includes an orientation notch in its top-right corner (obscured by camera angle), per the technique described in Section 3.5. Figure 6.5 shows two angles of the same artifact which were made using the mirrored model. The bottom of the artifact displays the same information as the top but reversed such that surfaces that are hard to reach on the top surface are easier accessible on the bottom and vice versa. This artifact also includes an orientation notch on its side (also obscured by camera angle).

Figure 6.6 shows an early printed artifact that was printed using some test data, which was created using random Gaussian sampling. However, this artifact failed mid-print as can be seen from the sudden cutoff. From the slight tilt near the top before the cutoff, we are led to believe that the individual pillars lacked enough support and started tilting to a point where the proposed toolpath differed too much from the current

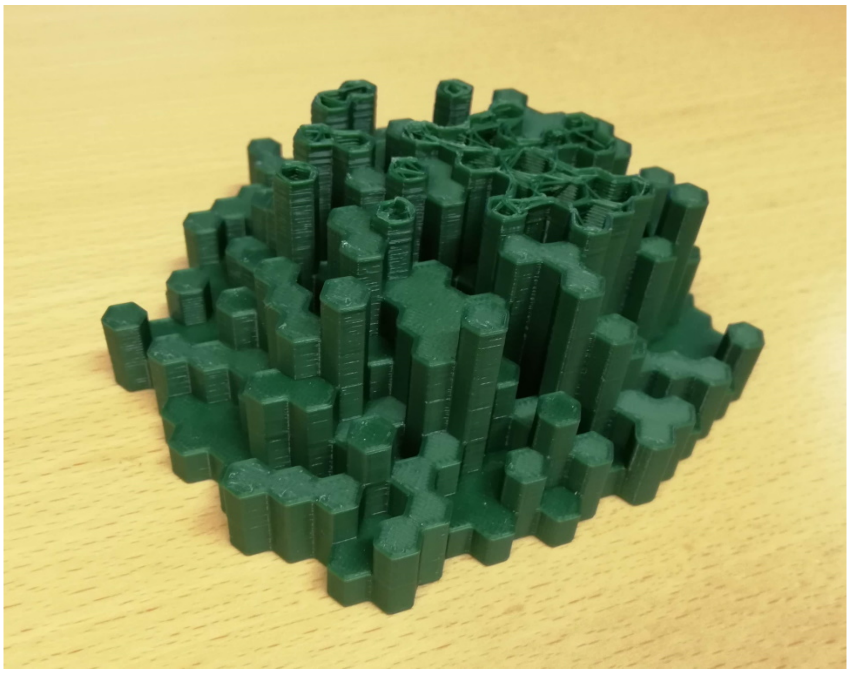


Figure 6.6: Model that failed during printing, likely due to lack of infill support

printed path and the print had to be canceled. In this print, we experimented with using a hexagonal infill pattern aligned with the hexagonal grid in the model, which, although it may look more artistically pleasing when the print gets cut off, usually is occluded by the model itself. Increasing the density of the infill pattern would increase the structural integrity of the artifact during the printing, ensuring higher success. And, since the infill pattern is invisible after the model is printed, would not have many drawbacks except for a slightly longer printing time, the resulting artifact being slightly heavier and slightly more material being consumed in the printing process.

The two artifacts in Figure 6.4 were printed using clear PLA filaments to experiment with transparency in the print. However, as can be seen from the images, the models themselves are not very transparent. It can also be observed that the artifacts are more transparent in the thinner parts of the model. In lightning theory used in optics and computer graphics, *microfacet* models are usually used to describe how light scatters when colliding with objects [44, 45]. Microfaset models describe how light statistically interacts with surfaces on a micro-level to explain how a surface visually looks. Matte surfaces have frequent variations on a micro-level which leads to little light being scattered in the same direction. Similarly, shiny surfaces have few variations on a micro-level, leading to more light being uniformly scattered. By the same logic, in our printed artifacts, we can imagine that a lot of light gets scattered due to uneven surfaces and the more material light passes through, the more opaque it looks. Printing the model with a thinner

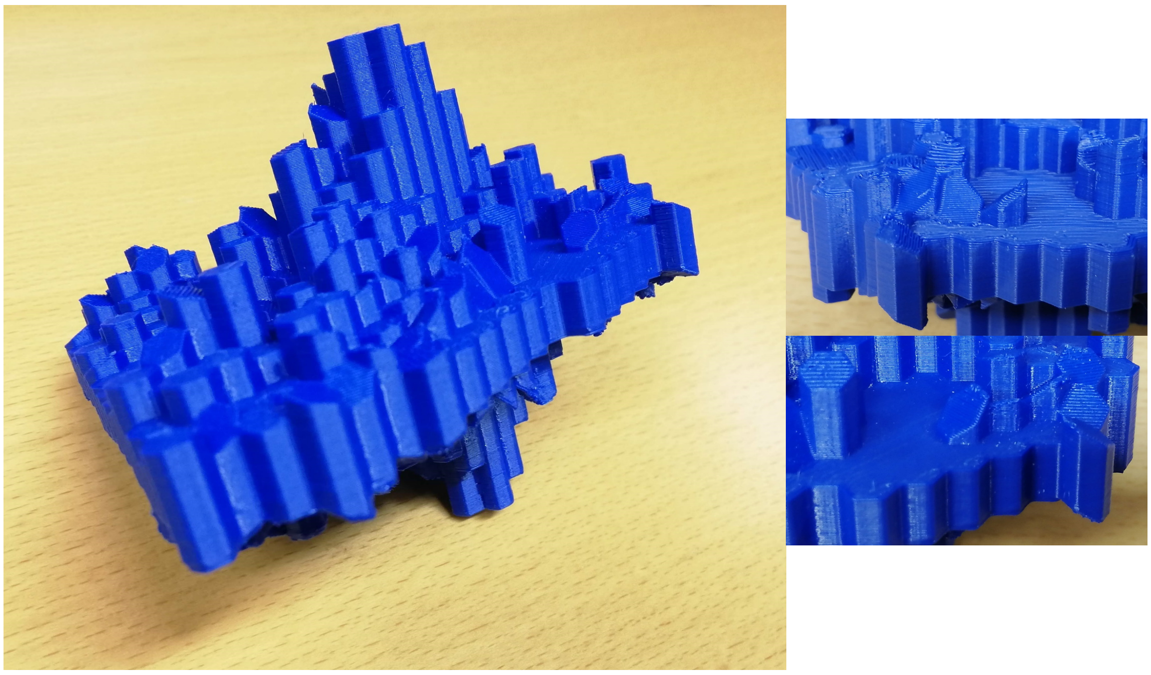


Figure 6.7: Model that failed due to early design flaws. The bottom part is an exact mirror of the top part, encoding the same data twice. Data will therefore be read wrong on the bottom part as the x and y axes will be inverted, which is *misleading* [120]. However, the regression plane encodings are not mirrored, making the bottom encode the exact opposite of its regression plane which is also *misleading*. The right images show a closeup of a regression plane encoding from each side.

material extrusion could also help make it look more transparent, and giving it less infill could also help in this regard. Using an infill pattern that would be aligned with the grid could possibly help with transparency from certain angles. Additionally, using 100% infill instead could also help with transparency as there is less room for scattering within the artifact. There are also techniques for further smoothing the outside of the model, for instance using an acetone treatment.

Figure 6.7 also displays a failed printed artifact using the airport dataset. However, unlike the artifact that failed due to printing, this artifact features a flawed design. In an early experiment to fix the surface cavities described in Section 3.4 we experimented with directly mirroring the top surface to the bottom. As the bottom surface is a direct mirror of the top this does not fix the issue with the surface cavities, it just reflects the same problem on the bottom surface as well. However, the real issue with the encoding was that the surfaces that were tilted to be aligned with the regression plane of the underlying data points, as described in Section 3.3, are not mirrored on the bottom surface and are instead tilted in the same direction. The bottom surface is a mirrored representation of the dataset, but the tilted surfaces also encode the opposite of the regression plane, giving two different encodings of the same data. How the encodings differ is displayed in the images to the right in Figure 6.7. From the images, it can be noted how the plane tilts toward its neighbor in the top image, but away from the same neighbor in the bottom image.

The implementation described in Chapter 5 was run on a laptop using an Intel i7-7700 CPU and a GTX 1070. For the scalar field that was generated and used in the above results, it took around 2 or 3 frames (30-50ms) from tweaking a parameter until the resulting geometry finished generating and a visual representation was displayed, while the program ran at a constant 60 frames per second.

6.2 Haptic Rendering User Study

To evaluate the effectiveness of the haptic rendering technique we employed a user study with 7 participants. None of the participants were classified as visually impaired and usually rely on vision as their primary sense for perception. Because of our participant's capabilities, the primary objective of the user study was to evaluate how well the participants could perform tasks without using their vision. If the participants who usually rely on vision for sensing environments can perform user tasks without much difficulty, a user more heavily relying on the sense of touch should have an easier time performing the same task as they are usually less dependent on vision.

In the user study, each participant was first introduced to the hardware, the virtual environment and a small selection of tools and their usages. Participants could move the device around the virtual environment and collide with surfaces to feel the forces pushing back; Additionally, the buttons on the haptic device enabled participants to switch between different levels of detail and toggle between the volume and the normal representation of the surface.

After participants had tested out the device and gotten used to the controls, the participants explored 4 constructed datasets, which are displayed in Figure 6.9, using the device. For each dataset, the participants were tasked with determining the highest and the lowest point, as well as describing how the surface was perceived. The participant would first explore the surface and complete the tasks with their eyes closed. But, after feeling satisfied with their answers to tasks, could open their eyes and compare their perceived surface with a visual representation of the surface.

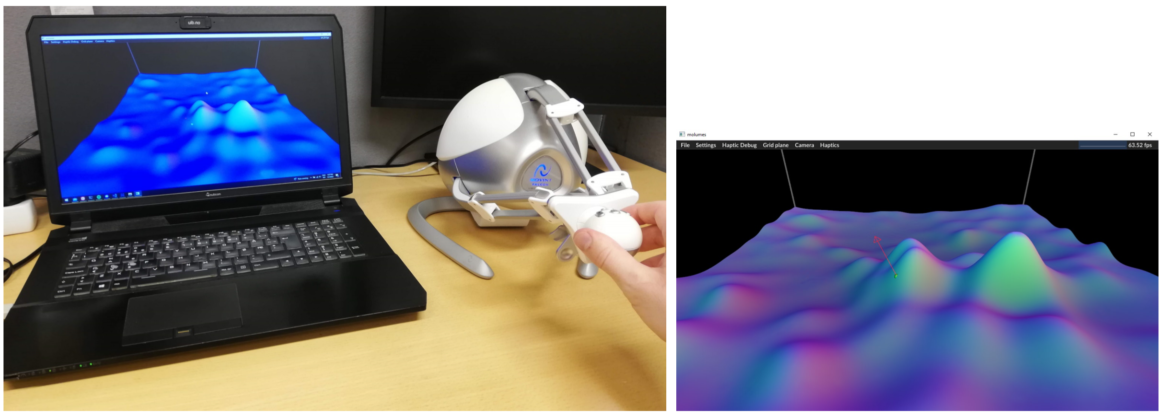


Figure 6.8: The setup for our user study.

6.2.1 Observations

P1 and P3 both expressed they had difficulties forming a mental image of the surface. A lot of the time was spent moving through the air instead of touching the surface and the highest and lowest points chosen were inaccurate. In both cases, we suspect the participants to not have received enough time to get used to the device before performing the task, which was also mentioned by P1. P1's results got more accurate throughout the study, further emphasizing this, even ending up describing the last surface as "bumpy". P3 had a harder time describing the surfaces but suggested this was caused by them usually having a hard time constructing mental images; even so, P3 managed to describe the last surface as being "wavy".

P5 also had difficulties getting an overview of the surface, but contrary to P1 and P3 had more time to get used to the device. Interestingly, P5 explored the surface a lot by repeatedly moving up in the air and down onto the surface, essentially "probing" the surface. Consequently, P5 results were more accurate as the surface was hit more often. While P5's highest and lowest points were not too accurate, P5's description of the surfaces was more correct compared to P1 and P3.

P1, P3 and P5 chose many of their highest points in the air, which we believe might have been caused by the participants perceiving the physical motor constraints of the device as part of the surface.

P2's, P4's and P7's results were the most accurate to the constructed data. Compared to the other participants, all of these participants explored the surface with much larger motions of the haptic device, rapidly moving back and forth. P7 moved slowly in the beginning but expressed the peaks were easier to detect with faster movements. P4's movement was mostly restricted to vertical movements in front of the device, while P2 moved more towards and away from the device. P7 also expressed they felt the spheres in the first dataset felt longer in the inwards direction of the device. We believe that this might have been caused by the device itself giving more resistance to inwards pushes due to its motor placements, making the spheres feel elliptical.

P2 and P4 identified and described more features of the surface compared to the other participants. On the third dataset, P2 successfully managed to identify a valley and distinguish the steepness of two slopes. P4 found the second dataset to be pretty much how they expected after having it revealed. P2 and P7 described the fourth surface as "wavy". P2 described it as consisting of around 10 slopes uniformly aligned in a grid, while P7 felt the surface mimicked ripples in water. Interestingly, P4 found the fourth dataset to be the most confusing which was contrary to the other participants.

P4 and P7 expressed that the bounds of the haptic device felt limiting in that some bumps did not even out before reaching the end of the bounds, which sometimes made them feel confused about whether the slope would continue beyond their limited view of the virtual environment. This suggests that the dataset used in the study should have been properly scaled down to fit within the bounds of the haptic device.

The accuracy of P6's results was somewhere in the middle of the other participants. Similar to P2, P4 and P7, P6 was moving a lot across the surface but had difficulties orienting themselves and the relative locations of structures in the virtual environment. P7 expressed this could be caused by their limited ability to create mental images.

Common for all participants is that they only took a moment to get an overview of the surface as soon as they opened their eyes, and the surface felt easier to navigate while looking at a visual representation of their position in space.

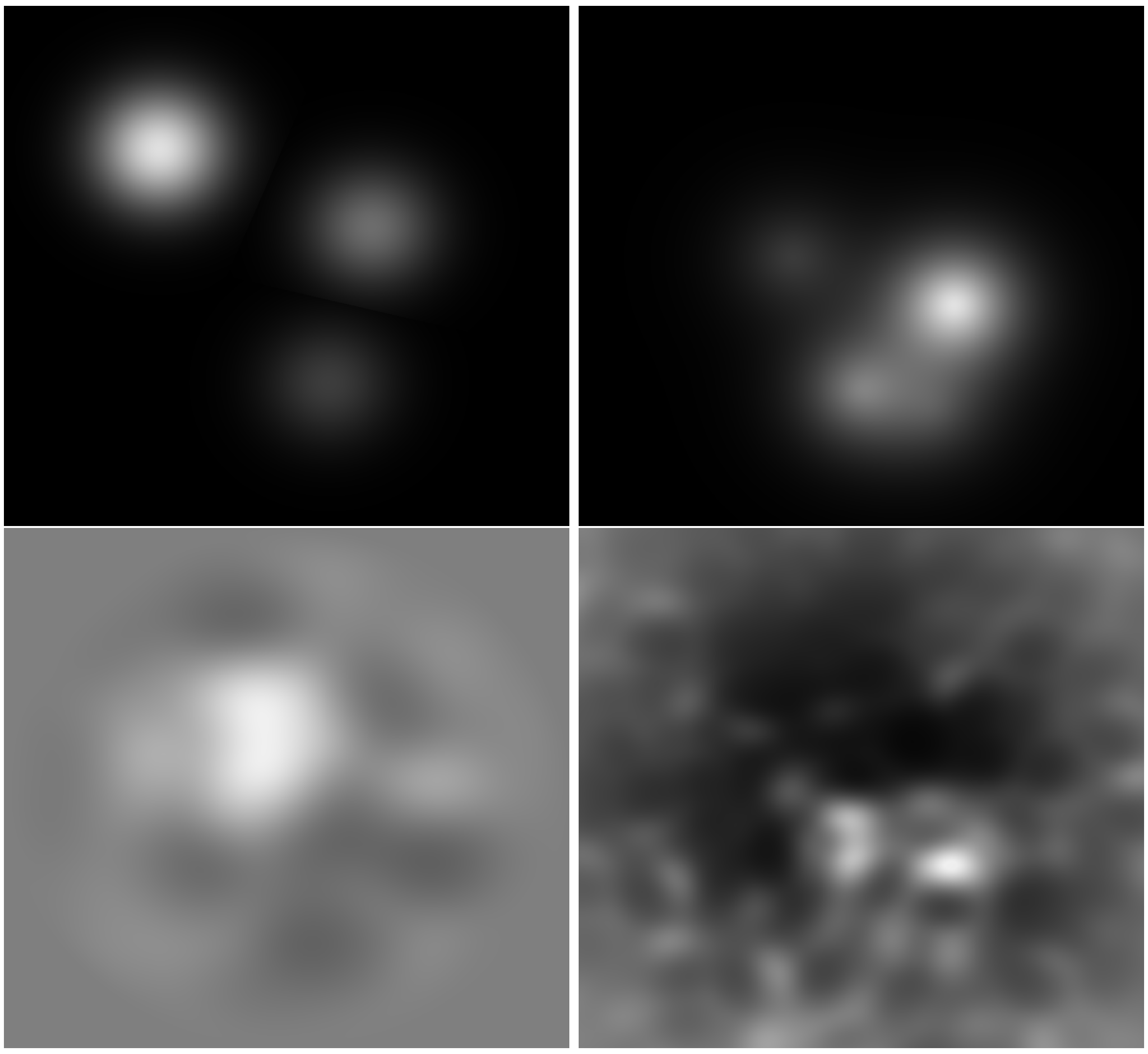


Figure 6.9: The heightfields used for the user study.

Chapter 7

Discussion

7.1 Physicalization

7.1.1 Reducing the Lie Factor

Currently, each data point from the scalar field represented by a tile is translated according to its value relative to the height of the model. This means the tile with the highest value in the scalar field is translated to be the same physical height as the model, while a tile with a zero value is translated to be leveled with other zero-valued tiles. While comparing data values with other data values to determine which one is higher and lower is easy, determining how much they differ, or fetching the value for a tile is much more difficult. To improve upon this we could encode each axis in a real-world measurement like centi- or milli-meters and encode a legend mark onto the model displaying the relationship between physically measured units and the value of the underlying data. However, this requires that we know the final dimensions the printed model will end up with. Unfortunately, this is not possible using the STL format as the format does not encode any units (see Section 2.3.2). Fortunately, the better specified 3MF format (version 1.2.3) does encode units into the file format meaning one can accurately control the printed model's dimensions.

This would solve the issue of reading relative values and differences, but because the base of our model is extruded an additional arbitrary amount to prevent zero-volume surfaces, values directly measured and read would have to subtract away the height of the base of the model to get the correct reading. As the extruded amount is chosen by our approach, a quick improvement would be to encode this value as a mark onto the model which users would then have to subtract from readings.

7.1.2 Improving Manufacturing Speed by Changing the Hull

In Section 3.2 we talked about how we utilized a convex hull to reduce the number of tiles that had to be included in our physicalization. However, in the proposed approach, every zero-valued tile inside the convex hull is still included in the 3D model. Not all zero-valued tiles contribute to keeping the surface connected, which means the extra tiles still cost a lot of unnecessary resources. More surface area printed both leads to more printing material used but also to a longer manufacturing process. To improve upon the resource cost it could be possible to instead limit the tiles included in the physicalization using a bounding concave hull, which is similar to a convex hull but encompasses its content by a tight-fitting boundary. One example of an algorithm that could solve this is the k-nearest neighbors algorithm presented by Moreira and Santos [132].

However, utilizing a tightly bounding concave hull creates a different issue. When the physicalization features plenty of zero-valued tiles, the similarity between them can help the user understand that they indicate a zero value and are not part of the encoded data. But if the bounding hull is as tight as possible, a few zero-valued tiles will connect separated *islands* instead, which may make it look like they represent data in the physicalization. This is especially true in cases where islands are separated by one or two tiles. The encoded non-existing data in the physicalization becomes whats referred to by Kindlmann and Scheidegger [120] as a *hallucinator*, meaning some part of the visualization does not represent data, instead it was "hallucinated" out of nothing but thin air. The problem of distinguishing which tiles are part of the physicalization and which are only there for structural purposes could be fixed by encoding the zero values with a different texture or shape, indicating that these tiles should not be grouped together with other data tiles.

7.1.3 Vertical Displacement From Surface Mirroring

Using the surface mirroring approach described in Section 3.4 works, but when physicalizing using input data with big changes in neighboring values, large displacements between the top and the bottom surfaces will be created. The large discontinuities in geometry might seem like it is a part of the visualization to a user, which, according to Kindlmann and Scheidegger [120], would be *misleading* as the big change in the physicalization represents an unimportant change in the data. One way of fixing this could be to use patterns or textures to indicate that they are part of the zero value, similar to how zero values were handled in Section 7.1.2.

However, it can be argued that the vertical displacement between the top and the bottom surface is not only there for structural purposes. Since the surface displacement is linearly dependent on the differences in scalar field values, the displacement encodes the magnitude of the biggest difference between neighboring scalar values, plus an additional small bias to ensure the mesh is hollow.

7.1.4 Orientation Triangle

We would additionally like to discuss one possible way to implement the alternative orientation mark described in Section 3.5. This mark is a right-angled triangle with its catheti aligned with the two axes of the hexagon grid. The problem with this mark, also mentioned in Section 3.5, is that to encode this mark we need to find a flat area large enough to fit it. As all of our hexagons are created in parallell, adding a mark on a single hexagon is trivial but adding a mark over multiple hexagons is not.

If the grid of hexagons is thought of as a graph with each tile as vertices and edges connecting all neighboring tiles on the same height, the *connected component* of a tile can be thought of as the set of all neighboring tiles on the same height. The largest connected component describes the largest set of connected tiles, but we need to find the connected component which can contain the largest 2D area within it to place our glyph. Additionally, since we want our glyph to represent the axes, our triangle has to be axis-aligned. However, we can easily fit a triangle in any direction inside a circle. Thus, finding the largest circle inside the connected component instead transforms our problem into the Maximum Inscribed Circle (MIC) problem, also known as the poles of inaccessibility problem.

Some solutions to the MIC problem has been written for MATLAB [133, 134] and a popular implementation for JavaScript exist [135] based on previous work by Garcia-Castellanos and Lombardo [136]. Beyhan et al. [137] also recently made a generalized approach based on the MATLAB extension by Birdal [133].

Our use case can be even further simplified because we only need to find one connected component which fits a circle large enough to fit our glyph. So if the size of the glyph is specified, our approach would be to greedily choose any connected component whose MIC is large enough to fit our glyph and then place our glyph in the center of the MIC.

7.2 Haptic Rendering

7.2.1 Reducing Confusion From Empty Space

The participants that spent a lot of time in the air in the user study had a hard time orienting themselves until they were allowed to open their eyes to see a visual representation of their position. We believe this to be caused by the lack of any forces being applied while moving through the air, possibly giving them too little information to create a mental image. Similarly, most participants found the last dataset to be the easiest to navigate around, possibly because of the high frequency of elevation changes. Fritz and Barner [117] argue that haptic sensing is very much a kinetic sense, and the lack of feedback while navigating through empty parts of the virtual environment therefore might have disrupted the creation of a mental image. A possible solution to this could have been to add a force that would pull you towards the center of the virtual environment, or use vibration to indicate the distance to the surface. It is worth noting that our volume technique described in Section 4.5, while toggleable, was disabled by default in the study. Performing a similar study but with the volume mode as the default could yield very different results.

Chapter 8

Conclusion

Most current visualization techniques are aimed at information transfer via the visual sense. But with more hardware support for alternative modalities and ways to perceive information, more work needs to explore how to efficiently convey data through our other senses. For instance, more work on how to best utilize our tactile sensory system will hugely benefit the visually impaired community, but can also help in the creation of better multimodal experiences.

Advances in AM lower the bar for rapid prototyping, further lowering the difficulty of creating passive physicalizations. In Section 1.2 it was highlighted how physicalization is a complicated process requiring expertise in both visualization and fabrication, and the need for more tools and pipelines to simplify the process is required.

In this thesis, we presented one such pipeline for manufacturing a passive physicalization of two-dimensional scalar field data. The presented approach builds an artifact that features a landscape of hexagonal pillars representing values as the pillar height. We explored some usability issues regarding the encoding format and proposed some improvements. One of the issues was the reachability of occluded tiles, which we improved by encoding the bottom of the artifact with the negated dataset. We had some ambiguity around representation orientation, which we improved by employing an orientation glyph and an orientation anchor encoding.

We presented one solution to encoding the surface of the pillars using tilt, but this encoding was limited to displaying a directional attribute, which we specifically used to display the point alignment within the aggregated scalar field used to create our artifact.

More work can be done on different usages of this encoding and other ways to encode additional attributes.

While we worked around our occlusion issue by adding an additional bottom encoding, future solutions could instead change the encoding to better work around the issue in the first place. For instance by changing the primitive type used for the grid or changing the projection of the physicalization layout, e.g. spherical, elliptical or perspective.

More work is put into haptic technology due to its promising possibilities in enhancing VR experiences, making haptic technology more attainable, more advanced and cheaper. Current work in active physicalization largely revolves around exploring new mediums of physicalizing data, but more work can be put into developing data physicalization techniques for existing technology. With the increasing availability of haptic technology, more active physicalization techniques need to utilize this technology. Little research has also been done on force-feedback haptic devices and their usages.

We have presented one approach to haptically render a two-dimensional scalar field using a force-feedback device. In our approach, we build a virtual environment and attempt to replicate the feeling of what moving across a similar environment in real life would feel like, by replicating normal, friction and gravity forces. As our approach simulates forces in real-time, the encoding and the data itself can be changed and the corresponding changes to the environment can be felt in real-time. We also explored an alternative method of navigating the same environment by constructing a volume of gradually smoothed surfaces and "guiding" the user through the volume using the gradient.

From our user study in Section 6.2, it was revealed that users were able to determine the shape of the surface, but to a varying degree. In Section 7.2.1 it was discussed how the lack of forces in midair could be one of the main reasons for our varying results. Future solutions should take this into account when developing a haptic virtual environment.

Due to the way we calculated our surface forces, some geometry was not possible to display. For instance, surface forces were limited to always point slightly upwards. Experimenting with different ways to project or bend the surface could possibly enable any direction of surface forces to work and possibly help reduce the surface slipperiness felt when moving across tops.

While both of our physicalizations were designed with the visually impaired as a target audience, we were unable to verify how well they worked with our target audience. It is possible that, as people that rely less on vision develop stronger perceptions of the other senses, our physicalizations are more efficient for our target audience.

Both our physicalizations increased dimensionality by visualizing 2D scalar data in 3 dimensions. The first one is by constructing geometry representable in 3 dimensions, and the second one by utilizing a three-dimensional display. Likewise, other popular visualization techniques that utilize dimensionality reduction to properly represent the data on a 2D computer screen or paper, can be visualized without reduction via physicalization in three dimensions.

Glossary

end-effector The physical part the user interacts with on a haptic force-feedback input device.

probe In the context of haptic simulation: The representation of the physical end-effector for the haptic device in the virtual environment.

slicer Software typically specific to a 3D printer that generates toolpath coordinates for the printer. The slicer software is responsible for slicing the geometry into layers that can be printed with AM.

List of Acronyms and Abbreviations

AM	Additive Manufacturing.
CSG	Constructive Solid Geometry.
DVR	Direct Volume Rendering.
FDM	Fused Deposition Modeling.
FFF	Fused Filament Fabrication.
GPGPU	General-Purpose computing on Graphics Processing Units.
MIC	Maximum Inscribed Circle.
VR	Virtual Reality.

Bibliography

[1] Tamara Munzner. *Visualization analysis & design.* Boca Raton, FL: CRC Press/-Taylor & Francis Group, 2015. ISBN: 9781466508934.

[2] *World report on vision.* Geneva: World Health Organization, 2019. ISBN: 9789241516570. URL: https://www.who.int/publications/i/item/9789241516570.

[3] Jinho Choi et al. "Visualizing for the Non-Visual: Enabling the Visually Impaired to Use Visualization". In: *Computer Graphics Forum* 38.3 (June 2019), pp. 249–260. DOI: 10.1111/cgf.13686.

[4] Shaun K. Kane and Jeffrey P. Bigham. "Tracking @stemxcomet: teaching programming to blind students via 3D printing, crisis management, and Twitter". In: *Proceedings of the 45th ACM technical symposium on Computer science education.* ACM, Mar. 2014. DOI: 10.1145/2538862.2538975.

[5] Leona Holloway, Kim Marriott, and Matthew Butler. "Accessiblemaps for the blind: Comparing 3D printed models with tactile graphics". In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems.* ACM, Apr. 2018. DOI: 10.1145/3173574.3173772.

[6] Simon Stusak, Jeannette Schwarz, and Andreas Butz. "Evaluating the Memorability of Physical Visualizations". In: *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems.* ACM, Apr. 2015. DOI: 10.1145/2702123.2702248.

[7] Yvonne Jansen, Pierre Dragicevic, and Jean-Daniel Fekete. "Evaluating the efficiency of physical visualizations". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.* ACM, Apr. 2013. DOI: 10.1145/2470654.2481359.

[8] H. Djavaherpour et al. "Data to Physicalization: A Survey of the Physical Rendering Process". In: *Computer Graphics Forum* 40.3 (June 2021), pp. 569–598. DOI: 10.1111/cgf.14330.

[9] Yvonne Jansen et al. “Opportunities and Challenges for Data Physicalization”. In: *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM, Apr. 2015. DOI: 10.1145/2702123.2702180.

[10] Saiganesh Swaminathan et al. “Supporting the design and fabrication of physical visualizations”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, Apr. 2014. DOI: 10.1145/2556288.2557310.

[11] Ricardo Sosa et al. “DATA OBJECTS: DESIGN PRINCIPLES FOR DATA PHYSICALISATION”. In: *Proceedings of the DESIGN 2018 15th International Design Conference*. Faculty of Mechanical Engineering and Naval Architecture, University of Zagreb, Croatia; The Design Society, Glasgow, UK, 2018. DOI: 10.21278/idc.2018.0125.

[12] Syed A.M. Tofail et al. “Additive manufacturing: scientific and technological challenges, market uptake and opportunities”. In: *Materials Today* 21.1 (Jan. 2018), pp. 22–37. DOI: 10.1016/j.mattod.2017.07.001.

[13] Bernhard Preim and Charl P. Botha. *Visual Computing for Medicine: Theory, Algorithms, and Applications*. 2nd ed. Morgan Kaufmann, 2014. 812 pp. ISBN: 9780124158733. URL: https://www.ebook.de/de/product/21020304/bernhard_preim_charl_p_botha_visual_computing_for_medicine_theory_algorithms_and_applications.html.

[14] Gourishetti Ravali and Muniyandi Manivannan. “Haptic Feedback in Needle Insertion Modeling and Simulation”. In: *IEEE Reviews in Biomedical Engineering* 10 (2017), pp. 63–77. DOI: 10.1109/rbme.2017.2706966.

[15] Cléber G. Corrêa et al. “Haptic interaction for needle insertion training in medical applications: The state-of-the-art”. In: *Medical Engineering & Physics* 63 (Jan. 2019), pp. 6–25. DOI: 10.1016/j.medengphy.2018.11.002.

[16] Timothy R. Coles, Dwight Meglan, and Nigel W. John. “The Role of Haptics in Medical Training Simulators: A Survey of the State of the Art”. In: *IEEE Transactions on Haptics* 4.1 (Jan. 2011), pp. 51–66. DOI: 10.1109/toh.2010.19.

[17] William Playfair. *The Commercial and Political Atlas*. 1786.

[18] David W. Scott. “A Note on Choice of Bivariate Histogram Bin Shape”. In: *Journal of Official Statistics* 4.1 (Mar. 1988). Copyright - Copyright Statistics Sweden (SCB) Mar 1988; Last updated - 2013-01-07, pp. 47–51. URL: https://www.proquest.com/scholarly-journals/note-on-choice-bivariate-histogram-bin-shape/docview/1266807897/se-2.

[19] Yan Holtz. *Where Surfers Live.* In: *data-to-viz.com.* https://github.com/holtzy/data_to_viz. 2022. URL: https://www.data-to-viz.com/story/GPSCoordWithoutValue.html (visited on 06/07/2022).

[20] D. B. Carr et al. "Scatterplot Matrix Techniques for Large N". In: *Journal of the American Statistical Association* 82.398 (June 1987), p. 424. DOI: 10.2307/2289444.

[21] William S. Cleveland and Robert McGill. "The Many Faces of a Scatterplot". In: *Journal of the American Statistical Association* 79.388 (Dec. 1984), pp. 807–822. DOI: 10.1080/01621459.1984.10477098.

[22] W. D. Dupont and Jr. Plummer W. D. "Using density-distribution sunflower plots to explore bivariate relationships in dense data". In: *Stata Journal* 5.3 (2005), 371–384(14). URL: https://www.stata-journal.com/article.html?article=gr0016.

[23] T. Trautner, M. Sbardellati, and S. Bruckner. "Honeycomb Plots: Enhancing Hexagonal Binning Plots with Spatialization Cues". In: *Eurographics Conference on Visualization (EuroVis) 2022* (2022). The paper is part of a current issue and not yet published.

[24] Martin Gronemann and Michael Jünger. "Drawing Clustered Graphs as Topographic Maps". In: *Graph Drawing.* Springer Berlin Heidelberg, 2013, pp. 426–438. DOI: 10.1007/978-3-642-36763-2_38.

[25] Marius Hogräfer, Magnus Heitzler, and Hans-Jörg Schulz. "The State of the Art in Map-Like Visualization". In: *Computer Graphics Forum* 39.3 (June 2020), pp. 647–674. DOI: 10.1111/cgf.14031.

[26] Gabor T. Herman and Hsun Kao Liu. "Three-dimensional display of human organs from computed tomograms". In: *Computer Graphics and Image Processing* 9.1 (Jan. 1979), pp. 1–21. DOI: 10.1016/0146-664x(79)90079-0.

[27] William J. Schroeder and Kenneth M. Martin. "Visualization Handbook". In: ed. by Charles D. Hansen, Jr. Johnson Chris R., and Chris R. Johnson. Elsevier Science & Technology, 2005. Chap. 1, p. 1061. ISBN: 9780080481647.

[28] Robert A. Drebin, Loren Carpenter, and Pat Hanrahan. "Volume rendering". In: *ACM SIGGRAPH Computer Graphics* 22.4 (Aug. 1988), pp. 65–74. DOI: 10.1145/378456.378484.

[29] William E. Lorensen and Harvey E. Cline. "Marching cubes: A high resolution 3D surface construction algorithm". In: *ACM SIGGRAPH Computer Graphics* 21.4 (Aug. 1987), pp. 163–169. DOI: 10.1145/37402.37422.

[30] Martin J. Dürst. “Letters: additional reference to marching cubes.” In: *Computer-Graphics*. Vol. 22. 5. Association for Computing Machinery (ACM), 1988, p. 243. DOI: 10.1145/378267.378271.

[31] G.M. Nielson and B. Hamann. “The asymptotic decider: resolving the ambiguity in marching cubes”. In: *Proceeding Visualization '91*. IEEE Comput. Soc. Press, 1991. DOI: 10.1109/visual.1991.175782.

[32] Timothy S. Newman and Hong Yi. “A survey of the marching cubes algorithm”. In: *Computers & Graphics* 30.5 (Oct. 2006), pp. 854–879. DOI: 10.1016/j.cag.2006.07.021.

[33] H. E. Cline et al. “Two algorithms for the three-dimensional reconstruction of tomograms”. In: *Medical Physics* 15.3 (May 1988), pp. 320–327. DOI: 10.1118/1.596225.

[34] Arie Kaufman and Klaus Mueller. “Visualization Handbook”. In: ed. by Charles D. Hansen, Jr. Johnson Chris R., and Chris R. Johnson. Elsevier Science & Technology, 2005. Chap. 7, p. 1061. ISBN: 9780080481647.

[35] Lisa Sobierajski et al. “A fast display method for volumetric data”. In: *The Visual Computer* 10.2 (Feb. 1993), pp. 116–124. DOI: 10.1007/bf01901947.

[36] Y. Livnat, Han-Wei Shen, and C.R. Johnson. “A near optimal isosurface extraction algorithm using the span space”. In: *IEEE Transactions on Visualization and Computer Graphics* 2.1 (Mar. 1996), pp. 73–84. DOI: 10.1109/2945.489388.

[37] M. Levoy. “Display of surfaces from volume data”. In: *IEEE Computer Graphics and Applications* 8.3 (May 1988), pp. 29–37. DOI: 10.1109/38.511.

[38] N. Max. “Optical models for direct volume rendering”. In: *IEEE Transactions on Visualization and Computer Graphics* 1.2 (June 1995), pp. 99–108. DOI: 10.1109/2945.468400.

[39] Randima Fernando. *GPU gems : programming techniques, tips, and tricks for real-time graphics*. Boston, MA: Addison-Wesley, 2004. ISBN: 0321228324.

[40] Gordon Kindlmann and James W. Durkin. “Semi-automatic generation of transfer functions for direct volume rendering”. In: *Proceedings of the 1998 IEEE symposium on Volume visualization - VVS '98*. ACM Press, 1998. DOI: 10.1145/288126.288167.

[41] Marc Levoy. “Efficient ray tracing of volume data”. In: *ACM Transactions on Graphics* 9.3 (July 1990), pp. 245–261. DOI: 10.1145/78964.78965.

[42] J. Kniss et al. “A model for volume lighting and modeling”. In: *IEEE Transactions on Visualization and Computer Graphics* 9.2 (Apr. 2003), pp. 150–162. DOI: 10.1109/tvcg.2003.1196003.

[43] J. Kniss, G. Kindlmann, and C. Hansen. “Multidimensional transfer functions for interactive volume rendering”. In: *IEEE Transactions on Visualization and Computer Graphics* 8.3 (July 2002), pp. 270–285. DOI: 10.1109/tvcg.2002.1021579.

[44] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory To Implementation*. Online version. 2021. URL: https://www.pbr-book.org (visited on 05/11/2022). See : Matt Pharr and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation. From Theory to Implementation*. Elsevier Science & Technology, 2004. ISBN: 9780123785800.

[45] Matt Pharr and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation. From Theory to Implementation*. Elsevier Science & Technology, 2004. ISBN: 9780123785800.

[46] L. C. Henyey and J. L. Greenstein. “Diffuse radiation in the Galaxy”. In: *The Astrophysical Journal* 93 (Jan. 1941), p. 70. DOI: 10.1086/144246.

[47] C. Heine et al. “A Survey of Topology-based Methods in Visualization”. In: *Computer Graphics Forum* 35.3 (June 2016), pp. 643–667. DOI: 10.1111/cgf.12933.

[48] Lin Yan et al. “Scalar Field Comparison with Topological Descriptors: Properties and Applications for Scientific Visualization”. In: *Computer Graphics Forum* 40.3 (June 2021), pp. 599–633. DOI: 10.1111/cgf.14331.

[49] Hamish Carr, Jack Snoeyink, and Ulrike Axen. “Computing contour trees in all dimensions”. In: *Computational Geometry* 24.2 (Feb. 2003), pp. 75–94. DOI: 10.1016/s0925-7721(02)00093-7.

[50] Gunther Weber, Peer-Timo Bremer, and Valerio Pascucci. “Topological Landscapes: A Terrain Metaphor for Scientific Data”. In: *IEEE Transactions on Visualization and Computer Graphics* 13.6 (Nov. 2007), pp. 1416–1423. DOI: 10.1109/tvcg.2007.70601.

[51] Patrick Oesterling et al. “Two-stage framework for a topology-based projection and visualization of classified document collections”. In: *2010 IEEE Symposium on Visual Analytics Science and Technology*. IEEE, Oct. 2010. DOI: 10.1109/vast.2010.5652940.

[52] I. Fujishiro et al. “Volume data mining using 3D field topology analysis”. In: *IEEE Computer Graphics and Applications* 20.5 (2000), pp. 46–51. DOI: 10.1109/38.865879.

[53] Shigeo Takahashi, Yuriko Takeshima, and Issei Fujishiro. “Topological volume skeletonization and its application to transfer function design”. In: *Graphical Models* 66.1 (Jan. 2004), pp. 24–49. DOI: 10.1016/j.gmod.2003.08.002.

[54] Gunther H. Weber et al. “Topology-Controlled Volume Rendering”. In: *IEEE Transactions on Visualization and Computer Graphics* 13.2 (Mar. 2007), pp. 330–341. DOI: 10.1109/tvcg.2007.47.

[55] D. M. Thomas and V. Natarajan. “Symmetry in Scalar Field Topology”. In: *IEEE Transactions on Visualization and Computer Graphics* 17.12 (Dec. 2011), pp. 2035–2044. DOI: 10.1109/tvcg.2011.236.

[56] Garrett C. Millar et al. “Tangible Landscape”. In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems.* ACM, Apr. 2018. DOI: 10.1145/3173574.3173954.

[57] Yutaka Tokuda et al. “MistForm”. In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems.* ACM, May 2017. DOI: 10.1145/3025453.3025608.

[58] Drew Hemment. “Emoto - visualising the online response to London 2012.” In: ISEA International; Australian Network for Art & Technology; University of Sydney, Jan. 1, 2013. URL: http://hdl.handle.net/2123/9644.

[59] Sean Follmer et al. “inFORM”. In: *Proceedings of the 26th annual ACM symposium on User interface software and technology.* ACM, Oct. 2013. DOI: 10.1145/2501988.2502032.

[60] Faisal Taher et al. “Exploring Interactions with Physically Dynamic Bar Charts”. In: *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems.* ACM, Apr. 2015. DOI: 10.1145/2702123.2702604.

[61] Daniel Rozin. *Mechanical Mirrors (Wooden Mirror).* 1999. URL: http://www.smoothware.com/danny/woodenmirror.html (visited on 05/18/2022).

[62] Daniel Leithinger and Hiroshi Ishii. “Relief”. In: *Proceedings of the fourth international conference on Tangible, embedded, and embodied interaction - TEI ’10.* ACM Press, 2010. DOI: 10.1145/1709886.1709928.

[63] Ivan Poupyrev et al. “Lumen”. In: *ACM SIGGRAPH 2004 Emerging technologies on - SIGGRAPH ’04.* ACM Press, 2004. DOI: 10.1145/1186155.1186173.

[64] S. K. Tang et al. *Tangible cityscape.* 2014. URL: https://tangible.media.mit.edu/project/tangible-cityscape/ (visited on 05/18/2022).

[65] Daniel Leithinger et al. "Direct and gestural interaction with relief". In: *Proceedings of the 24th annual ACM symposium on User interface software and technology - UIST '11.* ACM Press, 2011. DOI: 10.1145/2047196.2047268.

[66] Ken Nakagaki et al. "inFORCE". In: *Proceedings of the Thirteenth International Conference on Tangible, Embedded, and Embodied Interaction.* ACM, Mar. 2019. DOI: 10.1145/3294109.3295621.

[67] Majken K. Rasmussen et al. "Shape-changing interfaces". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.* ACM, May 2012. DOI: 10.1145/2207676.2207781.

[68] Olivier Bau, Uros Petrevski, and Wendy Mackay. "BubbleWrap". In: *CHI '09 Extended Abstracts on Human Factors in Computing Systems.* ACM, Apr. 2009. DOI: 10.1145/1520340.1520542.

[69] Chris Harrison and Scott E. Hudson. "Providing dynamically changeable physical buttons on a visual display". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.* ACM, Apr. 2009. DOI: 10.1145/1518701.1518749.

[70] Marcelo Coelho and Pattie Maes. "Sprout I/O". In: *Proceedings of the 2nd international conference on Tangible and embedded interaction - TEI '08.* ACM Press, 2008. DOI: 10.1145/1347390.1347440.

[71] ART+COM Studios. *BMW Museum: Kinetic Sculpture.* http://dataphys.org/list/bmw-kinetic-sculpture/. 2008. URL: https://artcom.de/en/project/kinetic-sculpture/ (visited on 05/19/2022).

[72] Themis Omirou et al. "Floating charts: Data plotting using free-floating acoustically levitated representations". In: *2016 IEEE Symposium on 3D User Interfaces (3DUI).* IEEE, Mar. 2016. DOI: 10.1109/3dui.2016.7460051.

[73] Wolf-Dieter Rase. "Creating physical 3D maps using rapid prototyping techniques". In: *True-3D in Cartography.* Springer, 2011, pp. 119–134.

[74] Pierre Dragicevic and Yvonne Jansen. *List of Physical Visualizations.* 2012. URL: http://www.dataphys.org/list (visited on 05/19/2022).

[75] Louis R. Nemzer. "Gravity Wave Spectrogram - LIGO". In: *List of Physical Visualizations.* Ed. by Pierre Dragicevic and Yvonne Jansen. 2017. URL: http://dataphys.org/list/gravity-wave-spectrogram-ligo/ (visited on 05/20/2022).

[76] GitHub. *Skyline.* 2022. URL: https://skyline.github.com (visited on 06/08/2022).

[77] Udayan Umapathi et al. “LaserStacker”. In: *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology.* ACM, Nov. 2015. DOI: 10.1145/2807442.2807512.

[78] Julien Gardan. “Additive manufacturing technologies: state of the art and trends”. In: *International Journal of Production Research* 54.10 (Nov. 2015), pp. 3118–3132. DOI: 10.1080/00207543.2015.1115909.

[79] A. García-Collado et al. “Advances in polymers based Multi-Material Additive-Manufacturing Techniques: State-of-art review on properties and applications”. In: *Additive Manufacturing* 50 (Feb. 2022), p. 102577. DOI: 10.1016/j.addma.2021.102577.

[80] Daehoon Han et al. “Rapid multi-material 3D printing with projection micro-stereolithography using dynamic fluidic control”. In: *Additive Manufacturing* 27 (May 2019), pp. 606–615. DOI: 10.1016/j.addma.2019.03.031.

[81] Md. Hazrat Ali, Nazim Mir-Nasiri, and Wai Lun Ko. “Multi-nozzle extrusion system for 3D printer and its control mechanism”. In: *The International Journal of Advanced Manufacturing Technology* 86.1-4 (Dec. 2015), pp. 999–1010. DOI: 10.1007/s00170-015-8205-9.

[82] Mohammad Abu Hasan Khondoker, Asad Asad, and Dan Sameoto. “Printing with mechanically interlocked extrudates using a custom bi-extruder for fused deposition modelling”. In: *Rapid Prototyping Journal* 24.6 (July 2018), pp. 921–934. DOI: 10.1108/rpj-03-2017-0046.

[83] Luquan Ren et al. “3D printing of materials with spatially non-linearly varying properties”. In: *Materials & Design* 156 (Oct. 2018), pp. 470–479. DOI: 10.1016/j.matdes.2018.07.012.

[84] Yan Li et al. “A Review on Functionally Graded Materials and Structures via Additive Manufacturing: From Multi-Scale Design to Versatile Functional Properties”. In: *Advanced Materials Technologies* 5.6 (Apr. 2020), p. 1900981. DOI: 10.1002/admt.201900981.

[85] Vahid Babaei et al. “Color contoning for 3D printing”. In: *ACM Transactions on Graphics* 36.4 (July 2017), pp. 1–15. DOI: 10.1145/3072959.3073605.

[86] Alan Brunton, Can Ates Arikan, and Philipp Urban. “Pushing the Limits of 3D Color Printing”. In: *ACM Transactions on Graphics* 35.1 (Dec. 2015), pp. 1–13. DOI: 10.1145/2832905.

[87] Philipp Urban et al. “Redefining A in RGBA”. In: *ACM Transactions on Graphics* 38.3 (June 2019), pp. 1–14. DOI: 10.1145/3319910.

[88] Alan Brunton et al. “3D printing spatially varying color and translucency”. In: *ACM Transactions on Graphics* 37.4 (Aug. 2018), pp. 1–13. DOI: 10.1145/3197517.3201349.

[89] Christoph Bader et al. “Making data matter: Voxel printing for the digital fabrication of data across scales and domains”. In: *Science Advances* 4.5 (May 2018). DOI: 10.1126/sciadv.aas8652.

[90] Mark A. Skylar-Scott et al. “Voxelated soft matter via multimaterial multinozzle 3D printing”. In: *Nature* 575.7782 (Nov. 2019), pp. 330–335. DOI: 10.1038/s41586-019-1736-8.

[91] M. A. S. R. Saadi et al. “Direct Ink Writing: A 3D Printing Technology for Diverse Materials”. In: *Advanced Materials* (Apr. 2022), p. 2108855. DOI: 10.1002/adma.202108855.

[92] Ali Mahdavi-Amiri, Philip Whittingham, and Faramarz Samavati. “Cover-It: An Interactive System for Covering 3d Prints”. In: GI ’15. Halifax, Nova Scotia, Canada: Canadian Information Processing Society, 2015, pp. 73–80. ISBN: 9780994786807.

[93] Yizhong Zhang et al. “Computational hydrographic printing”. In: *ACM Transactions on Graphics* 34.4 (July 2015), pp. 1–11. DOI: 10.1145/2766932.

[94] Christian Schüller et al. “Computational thermoforming”. In: *ACM Transactions on Graphics* 35.4 (July 2016), pp. 1–9. DOI: 10.1145/2897824.2925914.

[95] Kaufui V. Wong and Aldo Hernandez. “A Review of Additive Manufacturing”. In: *ISRN Mechanical Engineering* 2012 (Aug. 2012), pp. 1–10. DOI: 10.5402/2012/208760.

[96] Cătălin Iancu, Daniel Iancu, and Alin Stăncioiu. “FROM CAD MODEL TO 3D PRINT VIA “STL” FILE FORMAT”. In: *Fiabilitate şi Durabilitate* 1 (2010), pp. 73–80.

[97] Jonathan D. Hiller and Hod Lipson. *STL 2.0: A Proposal for a Universal Multi-Material Additive Manufacturing File Format.* 2009. DOI: 10.26153/TSW/15106.

[98] Prusa3D. *PrusaSlicer.* In: *GitHub repository.* 2022. URL: https://github.com/prusa3d/PrusaSlicer (visited on 05/25/2022).

[99] Slic3r. *Slic3r.* In: *GitHub repository.* 2022. URL: https://github.com/slic3r/Slic3r (visited on 05/25/2022).

[100] Ultimaker. *Cura*. In: *GitHub repository*. 2022. URL: https://github.com/Ultimaker/Cura (visited on 05/25/2022).

[101] 3MF Consortium. *3MF Specification*. May 25, 2022. URL: https://3mf.io/specification/.

[102] Sebastian Ullrich et al. "Virtual needle simulation with haptics for regional anaesthesia". In: Mar. 2010.

[103] HapticsHouse. *Novint's Falcon Haptic Device*. 2022. URL: https://hapticshouse.com/pages/novints-falcon-haptic-device (visited on 06/07/2022).

[104] William R. Provancher. *Creating Fully Immersive Virtual and Augmented Reality by EmulatingForce Feedback with Reactive Grip™ Touch Feedback*. Tech. rep. Union City, CA, USA: Tactical Haptics, Inc., Aug. 2019. URL: http://tacticalhaptics.com/files/TacticalHaptics_whitepaper_Aug2019-Rv2.pdf (visited on 05/29/2022).

[105] Inrak Choi et al. "Wolverine: A wearable haptic interface for grasping in virtual reality". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Oct. 2016. DOI: 10.1109/iros.2016.7759169.

[106] Hsin-Ruey Tsai, Yu-So Liao, and Chieh Tsai. "ImpactVest: Rendering Spatio-Temporal Multilevel Impact Force Feedback on Body in VR". In: *CHI Conference on Human Factors in Computing Systems*. ACM, Apr. 2022. DOI: 10.1145/3491102.3501971.

[107] Hong-Yu Chang et al. "FacePush: Introducing Normal Force on Face with Head-Mounted Displays". In: *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*. ACM, Oct. 2018. DOI: 10.1145/3242587.3242588.

[108] Vivian Shen, Craig Shultz, and Chris Harrison. "Mouth Haptics in VR using a Headset Ultrasound Phased Array". In: *CHI Conference on Human Factors in Computing Systems*. ACM, Apr. 2022. DOI: 10.1145/3491102.3501960.

[109] Ultraleap. *Haptics devices*. 2022. URL: https://www.ultraleap.com/haptics/ (visited on 06/07/2022).

[110] Ismo Rakkolainen et al. "A Survey of Mid-Air Ultrasound Haptics and Its Applications". In: *IEEE Transactions on Haptics* 14.1 (Jan. 2021), pp. 2–19. DOI: 10.1109/toh.2020.3018754.

[111] Malte Weiss et al. "FingerFlux". In: *Proceedings of the 24th annual ACM symposium on User interface software and technology - UIST '11*. ACM Press, 2011. DOI: 10.1145/2047196.2047277.

[112] Alaa Adel et al. “Rendering of Virtual Volumetric Shapes Using an Electromagnetic-Based Haptic Interface”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Oct. 2018. DOI: 10.1109/iros.2018.8593699.

[113] Shigeo Yoshida, Yuqian Sun, and Hideaki Kuzuoka. “PoCoPo: Handheld Pin-based Shape Display for Haptic Rendering in Virtual Reality”. In: *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. ACM, Apr. 2020. DOI: 10.1145/3313831.3376358.

[114] Ig Mo Koo et al. “Development of Soft-Actuator-Based Wearable Tactile Display”. In: *IEEE Transactions on Robotics* 24.3 (June 2008), pp. 549–558. DOI: 10.1109/tro.2008.921561.

[115] Yusuke Ujitoko et al. “Development of Finger-Mounted High-Density Pin-Array Haptic Display”. In: *IEEE Access* 8 (2020), pp. 145107–145114. DOI: 10.1109/access.2020.3015058.

[116] Catagay Basdogan, Chih-Hao Ho, and Mandayam A. Srinivasan. “A Ray-Based Haptic Rendering Technique for Displaying Shape and Texture of 3D Objects in Virtual Environments”. In: *Proceedings of the ASME DynamicSystems and Control Division*. Vol. 61. 1997, pp. 77–84.

[117] J.P. Fritz and K.E. Barner. “Design of a haptic data visualization system for people with visual impairments”. In: *IEEE Transactions on Rehabilitation Engineering* 7.3 (1999), pp. 372–384. DOI: 10.1109/86.788473.

[118] Seungmoon Choi and Hong Z. Tan. “Haptic rendering - beyond visual computing - Toward realistic haptic rendering of surface textures”. In: *IEEE Computer Graphics and Applications* 24.2 (Mar. 2004), pp. 40–47. DOI: 10.1109/mcg.2004.1274060.

[119] Joseph M. Romano and Katherine J. Kuchenbecker. “Creating Realistic Virtual Textures from Contact Acceleration Data”. In: *IEEE Transactions on Haptics* 5.2 (Apr. 2012), pp. 109–119. DOI: 10.1109/toh.2011.38.

[120] Gordon Kindlmann and Carlos Scheidegger. “An Algebraic Process for Visualization Design”. In: *IEEE Transactions on Visualization and Computer Graphics* 20.12 (Dec. 2014), pp. 2181–2190. DOI: 10.1109/tvcg.2014.2346325.

[121] Miguel Fernandez-Vicente et al. “Effect of Infill Parameters on Tensile Mechanical Behavior in Desktop 3D Printing”. In: *3D Printing and Additive Manufacturing* 3.3 (Sept. 2016), pp. 183–192. DOI: 10.1089/3dp.2015.0036.

[122] R.L. Graham. “An efficient algorith for determining the convex hull of a finite planar set”. In: *Information Processing Letters* 1.4 (June 1972), pp. 132–133. DOI: 10.1016/0020-0190(72)90045-2.

[123] Xiaoyuan Guo, Jun Xiao, and Ying Wang. “A survey on algorithms of hole filling in 3D surface reconstruction”. In: *The Visual Computer* 34.1 (Sept. 2016), pp. 93–103. DOI: 10.1007/s00371-016-1316-y.

[124] Emiliano Pérez et al. “A comparison of hole-filling methods in 3D”. eng. In: *International journal of applied mathematics and computer science* 26.4 (2016), pp. 885–903. ISSN: 2083-8492. DOI: 10.1515/amcs-2016-0063.

[125] Daniel Keim et al. “Visual Analytics: Definition, Process, and Challenges”. In: (Mar. 2008). DOI: 10.1007/978-3-540-70956-5_7.

[126] Meng Qi, Thanh-Tung Cao, and Tiow-Seng Tan. “Computing 2D constrained Delaunay triangulation using the GPU”. In: *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games - I3D ’12.* ACM Press, 2012. DOI: 10.1145/2159616.2159623.

[127] Inc. 3D Systems. *OpenHaptics Developer Software.* 2022. URL: https://www.3dsystems.com/haptics-devices/openhaptics (visited on 05/04/2022).

[128] F. Conti et al. “The CHAI libraries”. In: *Proceedings of Eurohaptics 2003.* Dublin, Ireland, 2003, pp. 496–500. URL: https://www.chai3d.org.

[129] Force Dimension. *Force Dimension SDK.* 2021. URL: https://www.forcedimension.com/software/sdk (visited on 05/04/2022).

[130] J.P. Ewins et al. “MIP-map level selection for texture mapping”. In: *IEEE Transactions on Visualization and Computer Graphics* 4.4 (1998), pp. 317–329. DOI: 10.1109/2945.765326.

[131] Arash Partow. *The Global Airport Database.* 2022. URL: http://www.partow.net/miscellaneous/airportdatabase/ (visited on 05/03/2022).

[132] Adriano Moreira and Maribel Yasmina Santos. “Concave hull: A k-nearest neighbours approach for the computation of the region occupied by a set of points”. In: *Proceedings of the Second International Conference on Computer Graphics Theory and Applications - Volume 2: GRAPP,* INSTICC. SciTePress, 2007, pp. 61–68. ISBN: 978-972-8865-71-9. DOI: 10.5220/0002080800610068.

[133] Tolga Birdal. *Maximum Inscribed Circle using Voronoi Diagram.* In: *MATLAB Central File Exchange.* 2011. URL: https://www.mathworks.com/matlabcentral/fileexchange/32543-maximum-inscribed-circle-using-voronoi-diagram (visited on 05/02/2022).

[134] Tolga Birdal. *Maximum Inscribed Circle using Distance Transform.* In: *MATLAB Central File Exchange.* 2011. URL: https://www.mathworks.com/matlabcentral/fileexchange/30805-maximum-inscribed-circle-using-distance-transform (visited on 05/02/2022).

[135] Mapbox. *Polylabel.* In: *GitHub repository.* 2022. URL: https://github.com/mapbox/polylabel (visited on 05/02/2022).

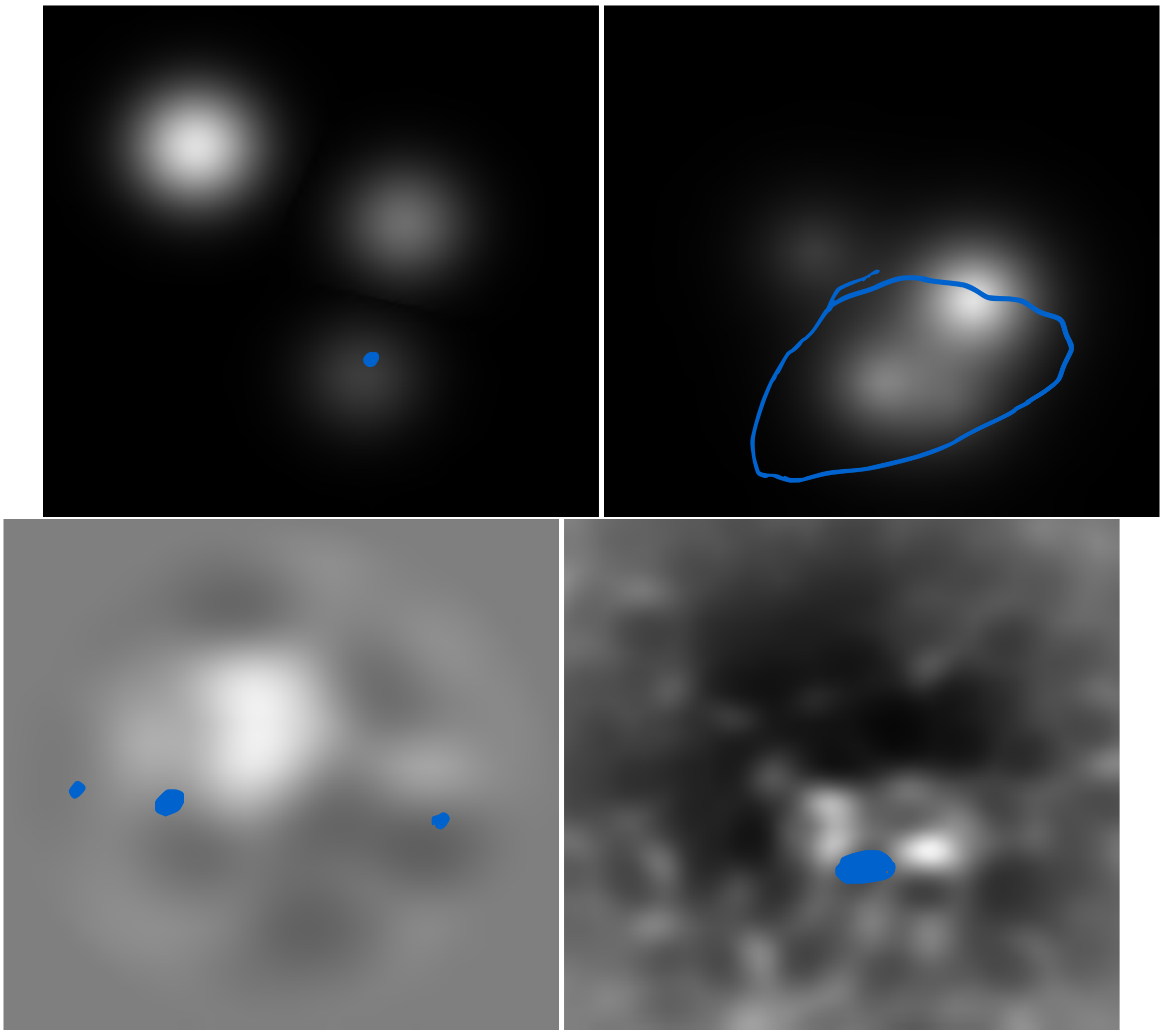
[136] Daniel Garcia-Castellanos and Umberto Lombardo. "Poles of inaccessibility: A calculation algorithm for the remotest places on earth". In: *Scottish Geographical Journal* 123.3 (Sept. 2007), pp. 227–233. DOI: 10.1080/14702540801897809.

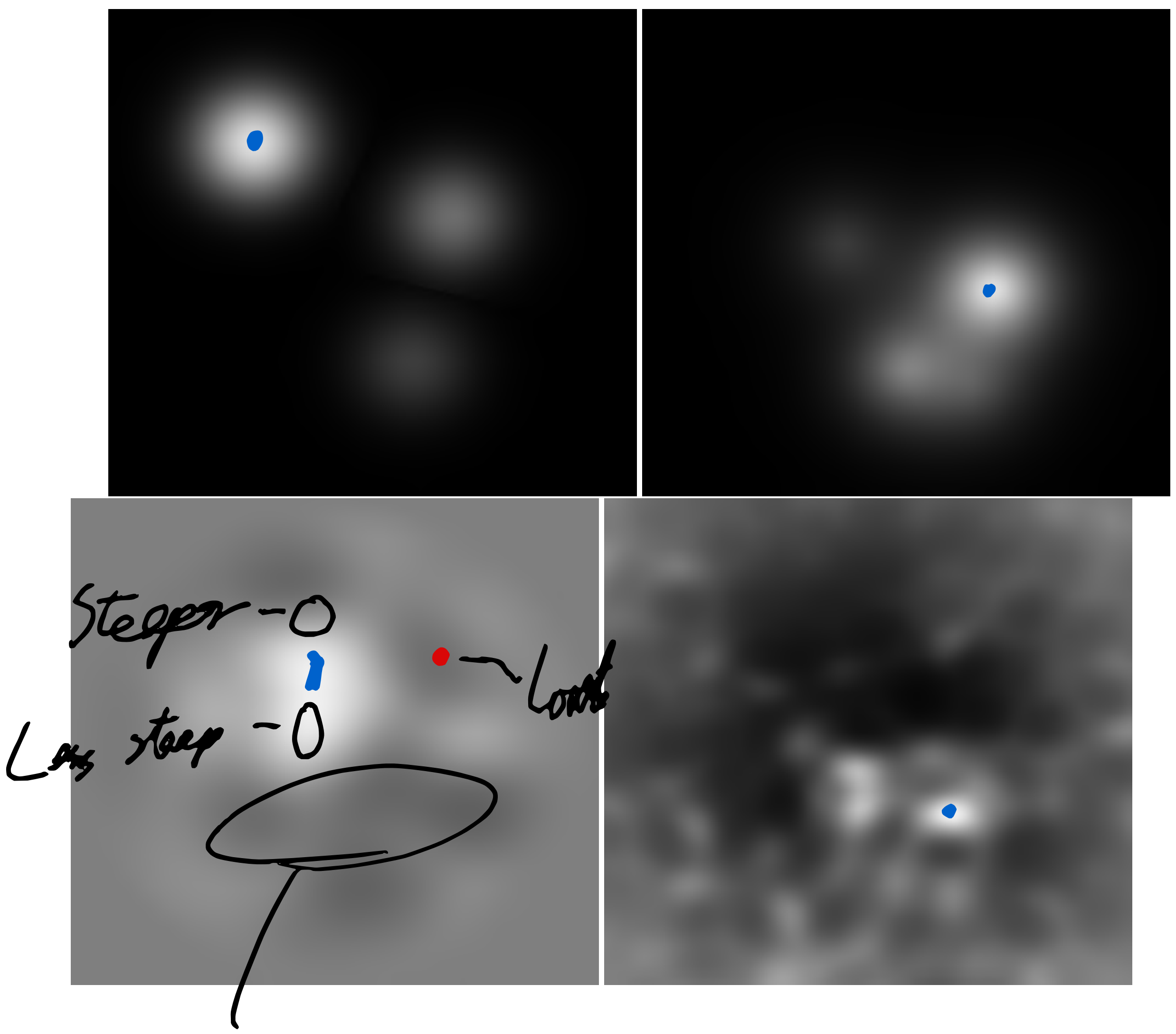
[137] Burak Beyhan, Cüneyt Güler, and Hidayet Tağa. "An algorithm for maximum inscribed circle based on Voronoi diagrams and geometrical properties". In: *Journal of Geographical Systems* 22.3 (May 2020), pp. 391–418. DOI: 10.1007/s10109-020-00325-3.

Appendix A

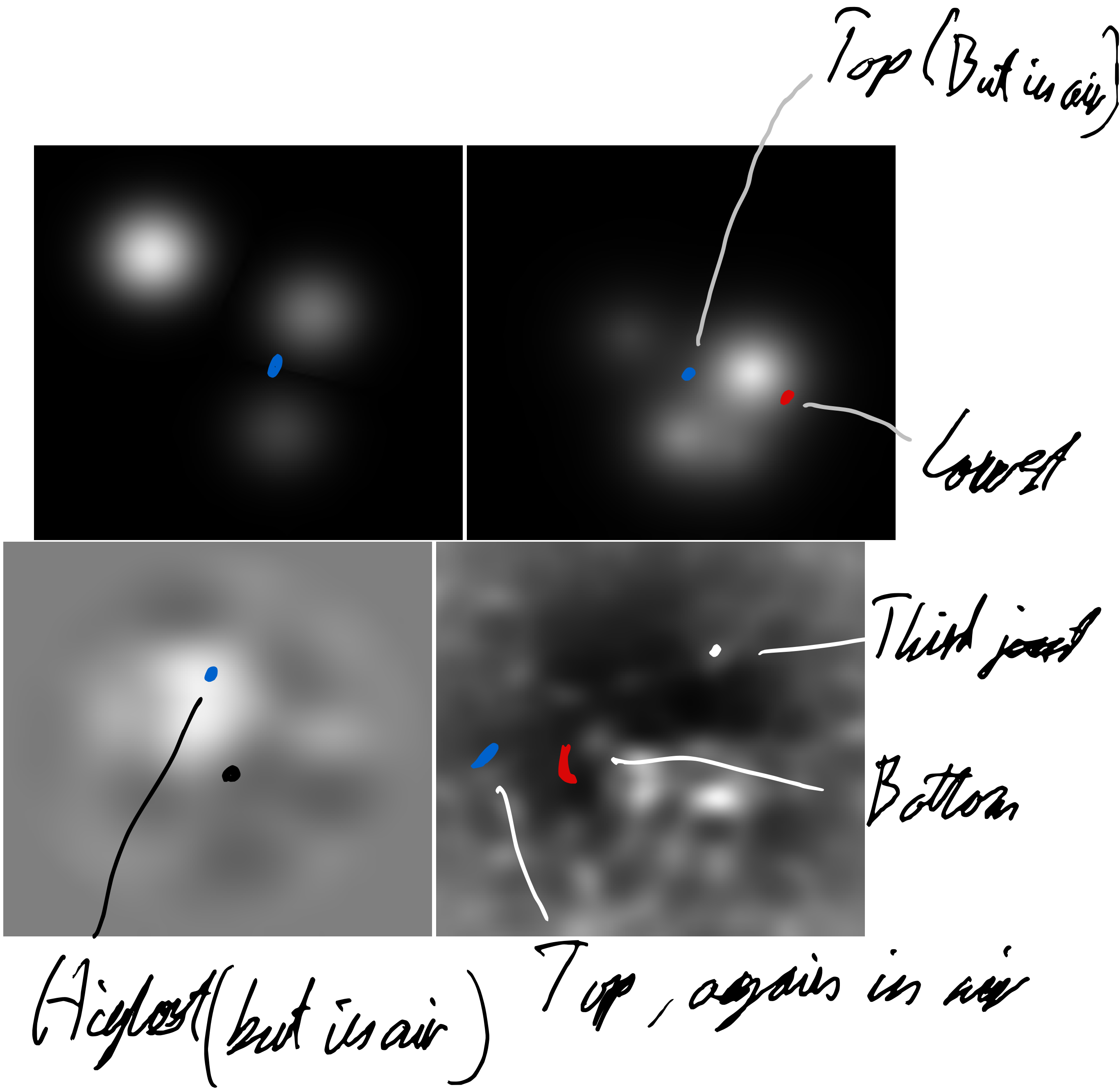
User Study Results

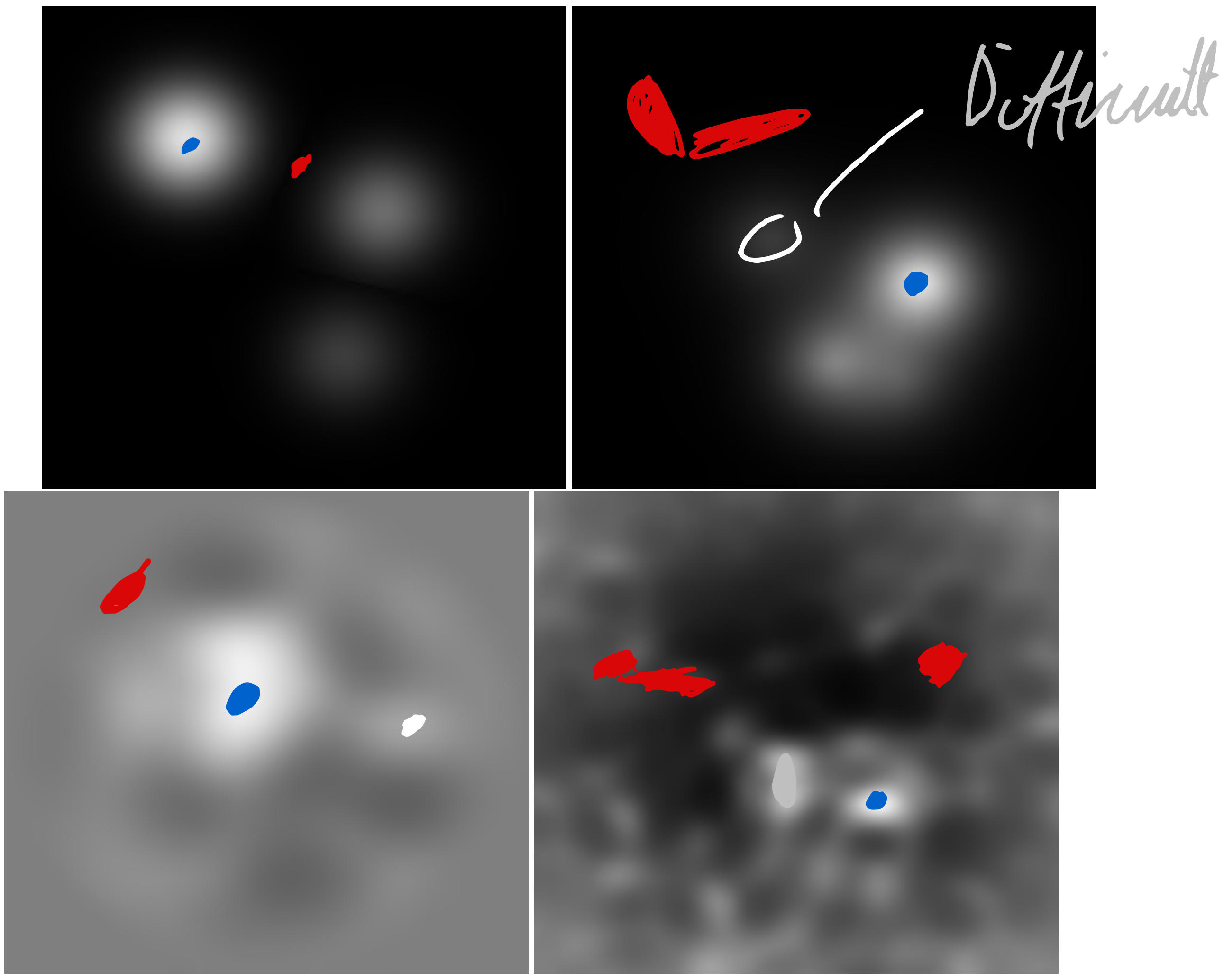
The results from the user study were manually recorded, which are displayed on the following pages, one page per participant. Blue marks the participant's maximum point, red marks the participant's lowest point or area and white or gray marks identified peaks or other areas of interest.



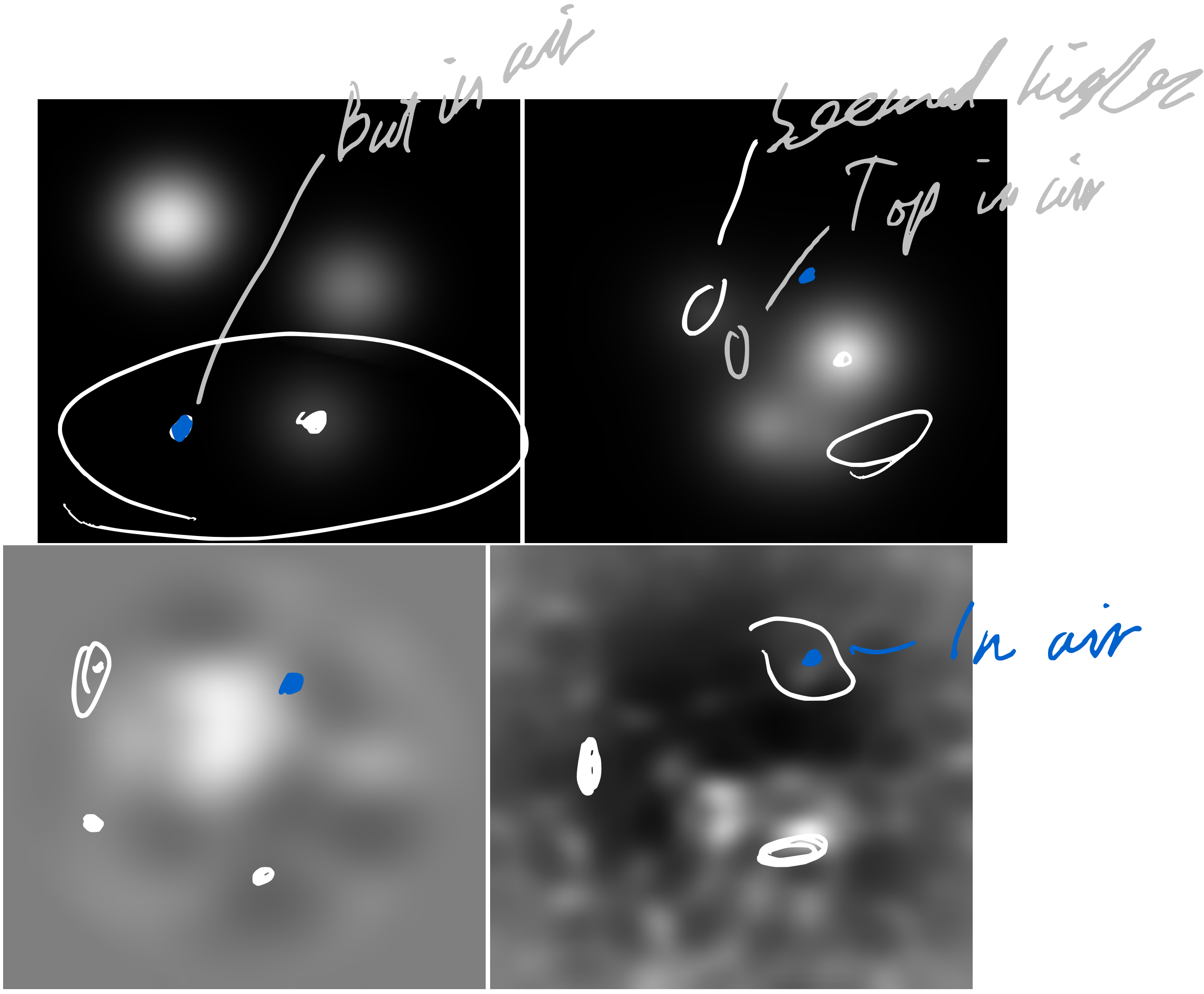


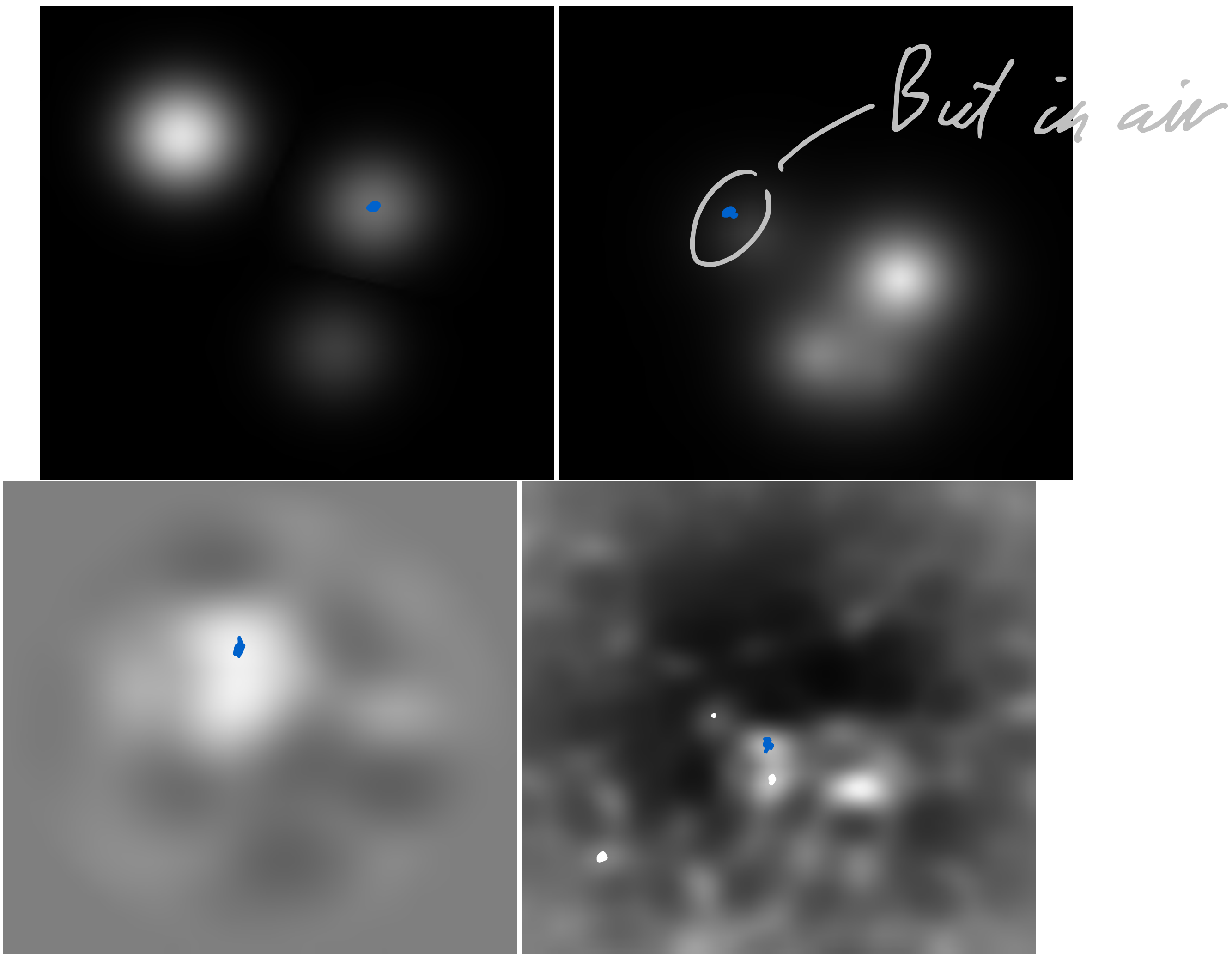
Identified as valley

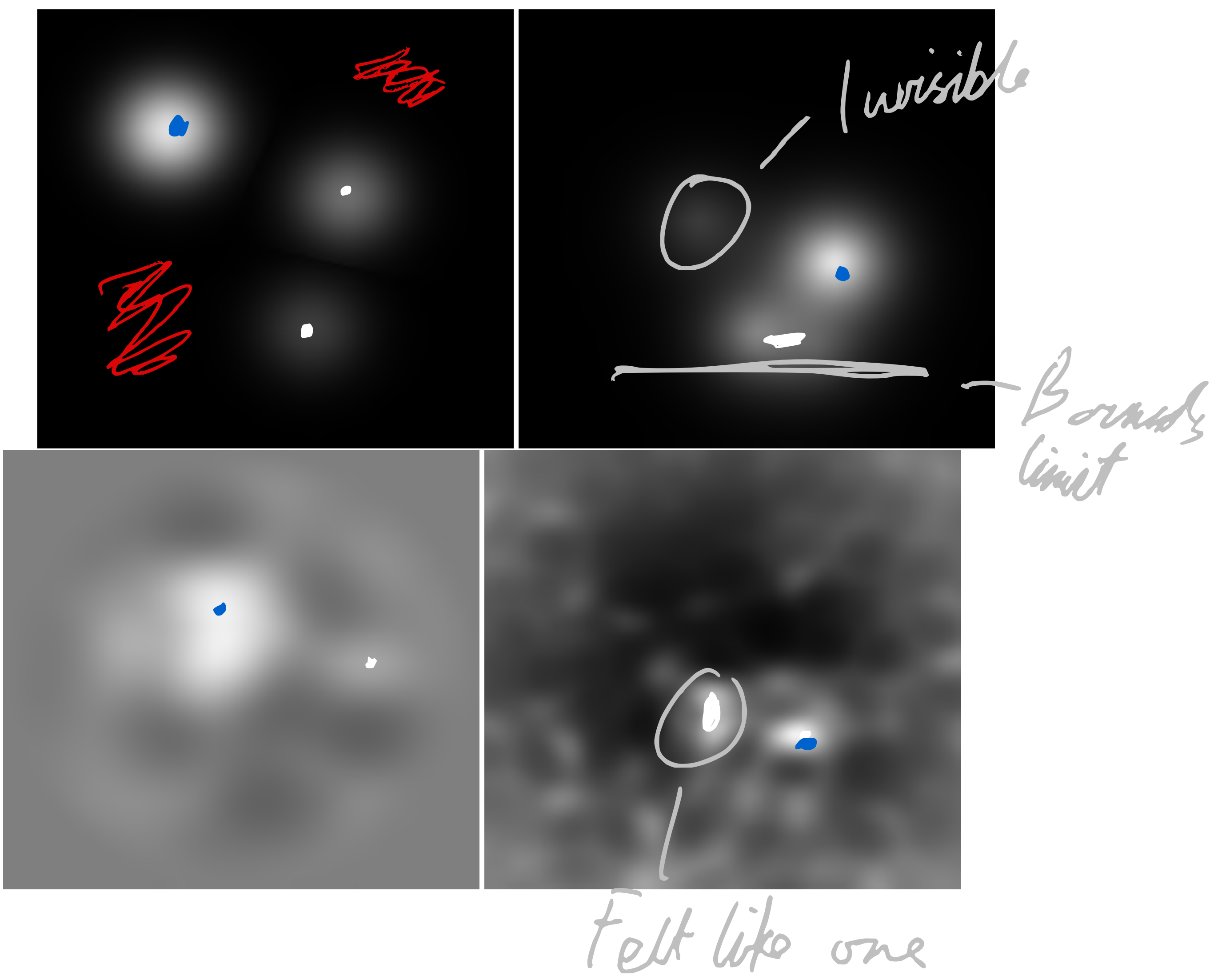

Top (But in air)
Lowest
Third joint
Bottom
Highest (but in air)
Top, again in air


Difficult

7




But in air


Invisible
Boundary limit
Felt like one