



universität
wien

DISSERTATION / DOCTORAL THESIS

Titel der Dissertation /Title of the Doctoral Thesis

„Slicing multi-dimensional spaces“

verfasst von / submitted by

Thomas Torsney-Weir

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of
Doktor der Technischen Wissenschaften (Dr. techn.)

Wien, 2018 / Vienna, 2018

Studienkennzahl lt. Studienblatt /
degree programme code as it appears on the student
record sheet:

A 786 880

Dissertationsgebiet lt. Studienblatt /
field of study as it appears on the student record sheet:

Informatik

Betreut von / Supervisor:

Univ.-Prof. Dr. Torsten Möller, PhD

To whomever finds this

Acknowledgements

I would like to acknowledge the invaluable support of my supervisor, Torsten Möller, as well as Michael Sedlmair. Their assistance and experience have been invaluable these last five years.

In addition, I want to thank all past and present members of the Visualization and Data Analysis research group at the University of Vienna. In particular, I want to thank Johanna Schlereth, Michael Phillips, Alireza Ghane, Raphael Sahann, Christoph Kralj, Bernhard Fröler, Mohsen Kamalzadeh, Hamid Younesy, Elena Rudkowsky, Patrick Wolf, Manfred Klaffenböck, Christoph Langer, Peter Ruch, Jennifer Prengel, Monika Gregor, Michaela Parzer, and Anne Marie Faisst. The discussions I've had over these past few years have been many and varied. I want to thank all of them for their suggestions and assistance in performing my research.

And last but certainly not least, I want to thank my family for their love, support, and encouragement throughout my life. Without their support I would not have left a lucrative but boring career in finance in order to become a happy student again. Furthermore, I would not have made it through the most difficult times of my PhD without their support.

Contents

List of Figures	viii
List of Tables	x
List of Algorithms	xi
1 Motivation	1
1.1 Multi-dimensional spaces	3
1.1.1 Manifolds	4
1.1.2 Shapes	5
1.2 Visualizing multi-dimensional continuous spaces	6
1.2.1 Encoding multi-dimensional data	7
1.2.2 Methods	8
1.3 Slices	9
1.4 Upcoming	11
2 1D slices	13
2.1 Motivation	14
2.2 Related Work	16
2.2.1 Discretization	16
2.2.2 Local methods	17
2.2.3 Global methods	18
2.3 Sliceplorer	18
2.3.1 Design requirements	19

2.3.2	Intuition	19
2.3.3	Focus point projection	20
2.3.4	Linked selection	21
2.3.5	Clustering	21
2.4	Task-based evaluation	22
2.4.1	Study design	24
2.4.2	Results	25
2.4.3	Summary	28
2.5	Usage scenarios	28
2.5.1	2D sinc function	29
2.5.2	Neural networks	30
2.5.3	Optimization algorithm	32
2.6	Discussion	34
2.7	Limitations and future work	35
2.8	Conclusion	36
3	2D slices	39
3.1	Motivation	40
3.2	Related work	42
3.2.1	Multi-objective optimization	42
3.2.2	Multi-dimensional objects	43
3.3	Algorithm	45
3.3.1	Conceptual overview	45
3.3.2	Algorithm details	47
3.4	Interface	50
3.4.1	Global view	50
3.4.2	Local view	51
3.5	Case studies	51
3.5.1	Polytopes	52
3.5.2	Positive and Bernstein polynomials	53

3.5.3	Pareto fronts	57
3.6	Algorithm performance	59
3.7	Conclusion and future work	61
4	Fast slicing	63
4.1	Motivation	63
4.2	Related work	66
4.2.1	Multi-D visualization	66
4.2.2	Multi-D interpolation	68
4.3	Problem description	69
4.3.1	1D analysis example	69
4.3.2	Applications in multi-D	72
4.3.3	Pipeline description	74
4.4	Requirements	75
4.4.1	Scene geometry	76
4.4.2	HyperSlice effect on scene geometry	77
4.4.3	Algorithm	78
4.5	Derivation of scene geometry	80
4.5.1	Filtering	80
4.5.2	Rendering	81
4.5.3	Expected total time	81
4.6	Fitting	83
4.6.1	Sampling	84
4.6.2	Final model	86
4.7	Timing results	86
4.7.1	Data fitting	86
4.7.2	Accuracy	88
4.8	Application scenarios	90
4.8.1	Constrain sampling	91
4.8.2	Subsample points	91

4.9	Limitations and future work	93
5	Conclusion	95
5.1	Future	96
5.2	Implications	97
	Bibliography	99
A	Full sliceplorer views	113
B	The expected number of points in a parameter space	115
B.1	Expected number of fragments	115
B.2	Expected 3D quad size	118
B.3	Expected quad size for the $d > 3$ case	119
B.3.1	Identities	121
B.3.2	Component derivation	123
B.3.3	Final derivation	125
C	Derivation of $\hat{Q}(r, d)$	127
C.1	Polar coordinates in n dimensions	129
C.2	Derivation	129
	Abstract	135
	Zusammenfassung	137

List of Figures

2.1	The evolution of Sliceplorer.	13
2.2	500 projected slices of the 5th dimension of the 5D Zakharov [7] function.	22
2.3	The four techniques we used to compare with 1D slices.	23
2.4	Different views of the 2D sinc function.	29
2.5	Different views of the predictions of four different machine learning regression models	31
2.6	1D slice and HyperSlice views of optimization algorithm traces	34
3.1	The interface for browsing slices created by the Hypersliceplorer algorithm.	39
3.2	An overview of the Hypersliceplorer algorithm	45
3.3	Views of regular polytopes	53
3.4	3-, 4-, and 5-dimensional hypercubes.	54
3.5	3-, 4-, and 5-dimensional hyperspheres.	54
3.6	Using Hypersliceplorer to examine differences in spaces	56
3.7	Differences in the space of general positive polynomials and Bernstein polynomials with positive Bernstein coefficients.	57
3.8	Hypersliceplorer views of spherical Pareto fronts in 3D and 5D.	57
3.9	Visualization of the 3-objective and 5-objective DLTZ1 problems in Hypersliceplorer.	58
3.10	Slicing operations per simplex	60
4.1	The function $f(x) = \sin(x) + \cos(2x)$ uniformly sampled with 10 points	70
4.2	Example of different interpretation methods	71

4.3	Gaussian process interpolation example	72
4.4	Tuner's HyperSlice view	75
4.5	Example of 2D slices of 2D and 3D spheres	78
4.6	Scatterplots of the time to render using the HyperSlice method.	87
4.7	Actual and relative error rates of prediction	89
4.8	Using rendering time prediction to constrain sampling	92
4.9	Interactive rendering times for various dimensions	93
B.1	Kernel/slice interaction	116

List of Tables

1.1	Rankings of visual encodings of quantitative data	8
2.1	Summary of the task-based evaluation	22
3.1	Timing results for Hypersliceplorer	59
4.1	A summary of the literature using Gaussian process models	73
4.2	Rendering time calibration results	89
4.3	Results of the cross-validation procedure.	91

List of Algorithms

1	Slicing a single simplex	49
2	Finding slices for all simplices	50
3	Rendering multi-dimensional data using HyperSlice and Gaussian process regression	79
4	Subsampling data to achieve interactive rendering time	93

1 | Motivation

We live in a three-dimensional world. Ourselves and what we can interact with are in three dimensions. We learn about our world by studying the various phenomena around us. These phenomena are described as continuous processes. In the beginning of our education we study function plots in high school. These give an intuitive view of one-dimensional phenomena. By exploring the relationship between an input factor and output, we can build an understanding on the relationship between the two. We can also compare one function plot to another. Visual inspection of these plots allows us to see common patterns. We use our pattern recognition ability to quickly categorize these different plots into different types of function behavior. Function plots can also be used to describe two-dimensional phenomena. These show the effect due to two input factors. In this case we can use the third dimension or color encoding to show the function value. From these plots we can also make general statements about the “shapes” of the behavior like how “peaky” the function is or if it is monotonically increasing. These shapes give us intuition into the underlying processes and help us learn about the world [88].

We also interact with many phenomena around us that are essentially multi-dimensional in nature. For example, the weather in a certain location is determined by the temperature, pressure, humidity, dew point, wind velocity, and wind direction, among others. A change in any of the factors results in a change in the weather. Each of these factors can be given a “spatial embedding.” Then, they can be viewed as a dimension of some space. By “walking” or navigating through this space we can observe the effect on the weather due to changes in these parameters.

Understanding multi-dimensional continuous spaces is difficult. As three-dimensional

beings we have real-world analogs for measurement, angle, and position in three dimensions. We do not have these once we move beyond three dimensions though. Nevertheless, visual analysis of these multi-dimensional spaces has produced insights about the underlying behavior [98, 38]. The issue is how to show more than three dimensions on a two-dimensional screen.

One strategy is to discretize the dataset through sampling and then use the wealth of discrete visualization tools available. Much of the work on multi-dimensional data analysis developed from analysis of tabular data. These datasets are often recorded from real-world events such as census, species, or text data and contain many different aspects about each entity. Thus, these datasets are inherently multi- or high-dimensional. Each different aspect of the data items creates a dimension to be analyzed. In the case of these data the mental model is that of discrete objects like humans, plants, or documents. Since the mental model is discrete in this case it makes sense to use data processing and visualization tools designed for discrete data. However, our mental model for physical data is a continuous one. Therefore, the discrete data paradigm breaks our mental model [116, 74]. Breaking the mental model means that the visualization is not conveying the complexities of the continuous phenomena. Rather, we should use visualization techniques purpose-built for continuous data.

While not as extensively developed, there is previous work on visualizing multi-dimensional, continuous data. These techniques can be broadly classified into projection, topological, and slicing methods. Projection methods attempt to distort the multi-dimensional object in order to view it on a two-dimensional screen. With projection techniques we can preserve distance, direction, size, or angles, but not all of these [106]. Depending on the projection method, we may see radically different representations. The issue is that it is not clear from the resulting visualization what sort of transformation was performed on the data. Thus it can be difficult to reconstruct the mental model of the multi-dimensional object. One of the most often seen multi-dimensional projection techniques is the Schlegel diagram [108] which picks a “face” of a polytope and projects the remaining faces inside it. Thus, this technique only works for 4D polytopes. Topological methods search the continuous dataset for

values of interest, such as critical points or contours. Topological visualization techniques also suffer from the issue of unclear transformation. It is difficult to relate the resulting visualization back to features in the multi-dimensional object.

Slice-based views of multi-dimensional continuous spaces have not been explored as extensively as other options. This work began with the advent of HyperSlice [119]. HyperSlice extends the idea of slicing from medical imaging to any number of dimensions. HyperSlice provides the framework for visualizing multi-dimensional continuous objects as a set of two-dimensional slices. There are $\binom{d}{2}$ subpanels, one for each pair of dimensions. Each panel shows a 2D slice of the object. The horizontal axis shows one dimension and the vertical axis shows another dimension. Essentially, each sub-plot of HyperSlice shows a 2D function plot. With 2D slices of solid multi-dimensional objects color is often used to encode value.

My work is inspired by the HyperSlice technique. Van Wijk and van Liere introduced the idea of using slice-based views of multi-dimensional data. However, they did not expand on what data types and tasks are involved in multi-dimensional continuous data analysis. I build on their work, investigating the usefulness of slice-based views of continuous multi-dimensional datasets. I also identified tasks involved in multi-dimensional data analysis. The task analysis informed the development of one- and two-dimensional slice-based views.

In this thesis I will explore the possibilities of these slice-based views. Through a number of case study examples, I will demonstrate the power of these views and ways to address their shortcomings.

1.1 Multi-dimensional spaces

There are a number of domains where one can apply the analysis of continuous multi-dimensional data. As of yet, there has not been a comprehensive data and task analysis for multi-dimensional continuous data analysis. For discrete data, there are several task analyses [103, 16, 3]. However, they are focused on identifying and selecting particular data items. Continuous datasets consist of ranges of values as well as func-

tions. Functions can be seen as a mapping from ranges of numbers to other ranges. The analysis task here is to study these ranges, their relationships to each other, and the mappings between them. Tasks addressing these have not yet been covered by visualization task analyses. Thus, there is no comprehensive source for what analysis tasks one wants to perform given a continuous multi-dimensional dataset. Work in this area has traditionally focused on developing a specific visualization for a specific task. For example, topological spines extracts critical points from a scalar field [24]. One goal of this thesis is to develop this task taxonomy for visualization of continuous multi-dimensional data.

These domains can be broadly classified into two types based on their analysis tasks. One type, *manifold* analysis, deals with understanding the relationship between inputs and outputs. This is a functional relationship. The user wants to inspect how changes in the inputs (independent variables) affect the outputs (dependent variables). One can also perform *shape* analysis. Here, in terms of the analysis tasks, there is no identification of independent and dependent variables. We look at each of these two in turn.

1.1.1 Manifolds

Studying the mapping between continuous ranges means studying functional relationships and thus manifolds. The critical issue is understanding the relationship between independent and dependent variables. Subtasks in manifold analysis include examining critical points, assessing the sensitivity of parameters, and understanding the shape of the manifold. One area where understanding the manifold is important is analyzing optimization surfaces and functions. In this case, the identification of extrema is important for understanding how many and the relative location of local optima. In addition, we want to understand the degree to which these are extrema. These can result in global optimization algorithms “getting stuck” in local optima rather than continuing to search for the global optimum. Optimization algorithms need to be carefully tuned to properly detect these features and ignore them where

necessary.

Simulations can be used to run experiments that are impractical or impossible in the real world. Simulation analysis is another area where the analysis tasks, in the abstract, are examining functions. If we look back at the weather simulation from before, the inputs to the function are things like the temperature and pressure. The output is, for example, the likelihood of rain the next day. The function is the simulation itself. Computer simulations are deterministic. A deterministic simulation has a fixed mapping from each unique input parameter configuration to an output value. This is the same as a functional relationship. The sensitivity and extrema are also important to simulation analysis. Thus, these can all be analyzed with visualizations of a manifold.

To date, manifold visualizations have concentrated on a particular analysis task or a particular application domain. For example, visualizations of the Morse–Smale complex [36] are focused on showing only critical points of the manifold. As with any visualization designed for a specific task, they must be used in combination with other views for visual analysis of domain-specific data. Domain-specific visualizations often used linked views to show different aspects of data to accomplish multiple tasks at once. However, they are purpose built for a specific domain. While techniques may transfer from one domain to another [99], it is not always clear how. My goal is to unify these methods to a certain extent. As I will show in chapter 2, slice-based views of manifolds can be used for a wide variety of tasks in a wide number of domains.

1.1.2 Shapes

One may also want to understand the relationship or correlation between multiple continuous values. In the manifold analysis case we have the notion of independent and dependent variables. This classification does not exist here. In this case we want to study the relationship of all variables. Careful study of the shape of the dataset can give insight into the relationship between the various ranges of dimensions of the object. For example, one may want to know if the overall shape is a sphere, donut, or box. In addition one may be interested in any kinks or cusps in the dataset. Changes in the

gradient and curvature of the shape are also of great interest. These indicate changes in correlation or relation.

The analysis tasks of multi-dimensional shapes can be applied to a number of different areas. The study of polytopes is one such area and perhaps the most direct application of understanding multi-dimensional continuous shapes. Polytopes are the multi-dimensional generalization of polyhedra and polygons. The tasks are to understand the symmetries and patterns making up the polytopes [126]. Perhaps a less obvious connection is the analysis of the tradeoff curves in multi-objective optimization. Since we are performing optimization we are interested in the tradeoffs amongst all the non-dominated points [67]. This is also known as the Pareto front. In this case, the user wants to understand what are the costs of reducing one or more parameters in order to increase the value of others. Cusps or large changes in curvature in these datasets are important since they show critical changes in the rate of tradeoff. With a proper view of a multi-dimensional object we can also view differences between two objects directly.

These two different data types and sets of tasks require different visualization considerations. Proper visualization for manifold analysis should focus on the relationships between independent and dependent variables. Visualizations of shapes do not have this mapping requirement and instead focus on the relationships between dimensions. With this categorization in mind, we can now examine the available visualization techniques to examine these.

1.2 Visualizing multi-dimensional continuous spaces

Understanding multi-dimensional space is difficult. As humans, we simply do not have the spatial analogs in more than three dimensions. A number of methods have been developed to extract specific features from the multi-dimensional object. For example, when studying polytopes, the number of faces and symmetries is very important [126].

However, these only produce an answer without sufficient context. They do not necessarily give any intuition as to how to transfer our three-dimensional knowledge to multi-dimensional spaces.

Visualizations of multi-dimensional spaces on a 2D screen must contend with some sort of reduction of the information. A proper visualization must select visual encodings that highlight the information we want to see. Any sort of data reduction requires trade-offs. The best visualization choices acknowledge any deficiencies to a particular visual encoding. By acknowledging these deficiencies, we can design tools to compensate for their shortcomings while still maintaining their advantages. Therefore, it is worth first looking at the possible mappings of data to visual elements. Then, I present commonly used visual encodings of multi-dimensional continuous data using these mappings.

1.2.1 Encoding multi-dimensional data

Multi-dimensional continuous data consists of a set of continuous ranges, one for each dimension. In the case of manifold analysis, each of these ranges can be additionally classified as “dependent” or “independent” depending on which side of the mapping they are on. Typical visualization practice is to give each dimension a separate visual channel. There are a number of possible visual channels that have been identified. The ranking of effectiveness of visual channels (shown in Table 1.1) was proposed by Bertin [12] and confirmed through experiments by Cleveland and McGill [23], Mackinlay [76], and Heer and Bostock [47]. Munzner [84] provides a summary of the results. We are also limited in how many channels we can use simultaneously. According to Ware [120], certain channels, such as red and green are not visually separable.

The difficulty of visualizing a continuous multi-dimensional space on a two-dimensional screen brings a number of challenges. We treat each dimension separately, thus, we need several different visual channels. However, there are simply not enough visual channels available to draw a 15-dimensional object in a single view. This is further

Table 1.1: Rankings of visual encodings of quantitative data

Bertin [12]	Cleveland and McGill [23]	Mackinlay [76]	Munzner [84]
Position	Position along a common scale	Position	Position on common scale
Size	Position along identical, nonaligned scales	Length	Position on unaligned scale
(Grey) Value	Length	Angle	Length (1D size)
Texture	Direction	Slope	Tilt/angle
Color	Angle	Area	Area (2D size)
Orientation	Area	Volume	Depth (3D position)
Shape	Volume	Density	Color luminance
	Curvature	Color saturation	Color saturation
	Densities	Color hue	Curvature
	Shading		Volume (3D size)
	Color saturation		

complicated by the fact that separate visual channels are not necessarily visually separable. Furthermore, each dimension of the multi-dimensional object under study is treated equally. For example, no particular axis of a polytope is more important than any other. We should encode each dimension using equally weighted effectiveness channels. With fewer channels available than data dimensions we either need to reduce the data or use multiple views to properly visualize the data.

1.2.2 Methods

The common taxonomy of how to view multi-dimensional data on screen is based on discrete data analysis. There, there are two categories: dimension reduction or projection. With continuous data, though there is a third possibility, that of slicing. Therefore, I view the taxonomy of *continuous* multi-dimensional data analysis methods into two categories. Data-driven methods include both projection and dimension reduction and reduce the dimensionality of the data before visualization. View-based methods reduce the data during the visualization. Slicing is a view-based method.

Purely data-driven methods are commonly known as feature selection or dimension reduction. The goal is to find a subset of dimensions that are critical to understanding the dataset. Topological techniques take this a step further. They discard all spatial information about the dataset and only concentrate on the difference in function value, as in the Morse-Smale complex [41], or evolution of contours, as in the contour tree [21]. Projections also synthesize the dataset into new dimensions to

show using visual channels. Principal component analysis [52] is a popular choice in this area. This rotates the space and thus produces new set of dimensions that are a linear combination of the input dimensions. Even this relatively simple operation (a rotation) can be difficult to understand. For example, iPCA [61] was a tool to help users understand the effects of the dimensional transformation.

View-based methods try and produce multiple linked views of a multi-dimensional dataset from different angles. Each view shows a subset of the dimensions. This way we can use a proper set of visual channels for each view. We use interaction to link these different views. These multiple, coordinated, linked views have been one of the biggest success stories from the visualization community [91]. The traditional HyperSlice [119] technique falls into this category. Each panel of the HyperSlice view shows two of the input dimensions and the value is encoded with color. The views are linked through the focus point selection. Changing the focus point in one sub-plot updates the other sub-plots.

My work focuses on the exploration of the combination of data-driven and view-based methods. Data-driven methods reduce the data in a way that we can get a global overview of the dataset. View-based methods are much more detailed but can only produce a local view of the data. By combining these methods I can achieve both a global overview as well as an on-demand local view of the dataset in a single visualization.

1.3 Slices

Slicing offers a number of advantages over other multi-dimensional visualization techniques. Slicing is a direct visualization of the multi-dimensional object. In contrast to methods like projection or dimension reduction, slicing does not distort the dimensions in order to display them on a two-dimensional screen. Since there is no distortion, distances in the visual representation are directly proportional to distances in the object. This is one of the reasons that slicing is popular in the medical imaging community. Sizes of organs or tumors can be measured visually on screen. Addi-

tionally, relative sizes correspond to what a doctor would expect to see in the body. Multi-dimensional data is abstract. It lacks the correspondence to real-world objects that the medical community uses to understand their two- and three-dimensional datasets. Nevertheless, it is still important in multi-dimensional data analysis to properly estimate distances and relative sizes.

Another benefit of the direct visualization is that users do not require extensive training to understand the visualization. The concept of slicing through a three-dimensional object is a familiar one. Humans are used to this even from slicing fruits and vegetables with a knife. This concept of slicing can be extended from this well-known metaphor to cover multiple slices of multi-dimensional objects.

Slice views use the horizontal and vertical axes for showing the effects due to the input parameters. These axes are the most perceptually uniform [109] and are considered the most effective (Table 1.1). One- or two-dimensional plots are replicated for each combination of dimensions in order to show more than two dimensions at once. This promotes familiarity of the visualization. Once the user has learned to read a single panel, they can apply this knowledge to the remaining panels. This approach follows the principal of small multiples [4].

In order to produce a slice plot one needs to first pick a particular *focus point* in the multi-dimensional space. This focus point determines which slices are being viewed. Selecting a good focus point a-priori is difficult. It either requires a great deal of luck or careful analysis of the dataset. This is not always possible. Slice-based views require some sort of interactive focus point selection. Interactively browsing through the slices requires interaction controls to give the user control over the focus point. Furthermore, we need some kind of navigation map to show which focus points the user has selected so that they do not become lost. Neither of these navigation aids are well developed at this point. This need for interaction is likely one of the reasons that slice-based views have not developed as much as projection or topological techniques. Static views are much easier to include in papers and don't require explanation prior to use.

The other implementation issue of slice-based views is ensuring that the visual-

ization can remain interactive. In this case, interactive is defined as the rate at which the user can maintain their concentration [101]. This is often defined at 10 frames per second. It can be difficult to compute a 2D slice of an arbitrary complex multi-dimensional object.

1.4 Upcoming

My goal is to highlight the advantages of slice-based methods for multi-dimensional data analysis. At the same time I want to address the limitations. The end goal is to bring slice-based views into the standard toolbox of visualizations.

In chapter 2, I examine how to visualize manifolds using slices. I present Sliceplorer which is a system to view one-dimensional slices of multi-dimensional manifolds. I use projections of these one-dimensional slices instead of showing one focus point at a time. I also go into detail about the tasks one wants to perform with manifold analysis.

I extend the idea of projections of slices to the second dimension in chapter 3. I show how this can be used to effectively visually analyze the shape of multi-dimensional data. In many cases, this data is given as a simplicial mesh. I introduce an algorithm to compute 2D slices of this mesh. Using this algorithm, I show how we can visually understand datasets like Pareto fronts or polytopes.

Finally, in chapter 4, I discuss how we can take advantage of the multi-dimensional geometry and GPU architecture to allow interactive-speed browsing of the focus points. In addition to this algorithm, I develop a method that can predict the amount of time needed per element to draw one frame of the visualization. I then show how this estimation formula can be calibrated to a particular user's hardware.

2 | 1D slices

First, we turn to the visual exploration of multi-dimensional continuous functions. In this case, the user is primarily interested in the relationship between dependent and independent variables. In this chapter, I discuss how one-dimensional slices can be a highly intuitive way of visualizing multi-dimensional manifolds. I also discuss how we can use projections of slices to address the focus point selection issue. The task analysis in this chapter introduces manifold analysis tasks. This chapter was based on work published as Thomas Torsney-Weir, Michael Sedlmair, and Torsten Möller. Sliceplorer: 1D slices for multi-dimensional continuous functions. *Computer Graphics Forum*, 36(3):167–177, June 2017.

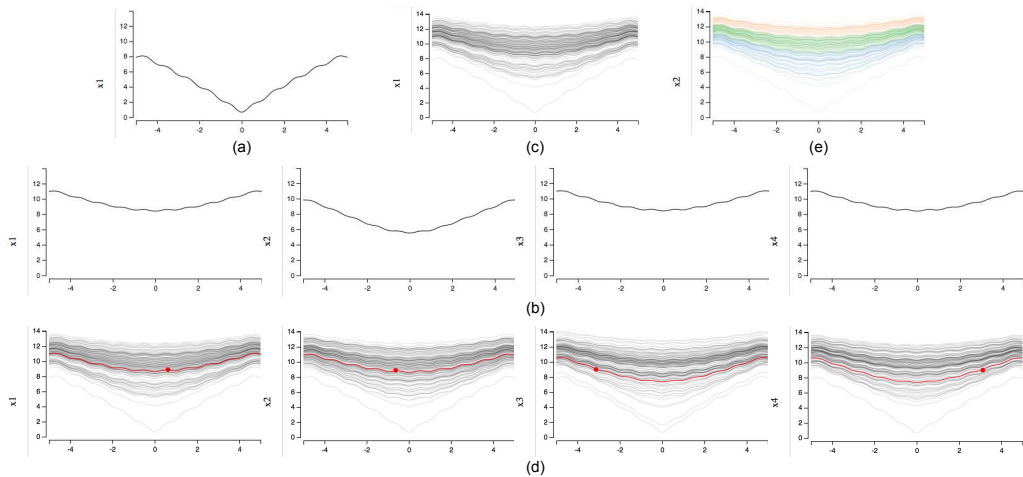


Figure 2.1: The evolution of Sliceplorer. I adapt the commonly known technique of 1D function plots (a) to multiple dimensions by taking a small multiples approach and repeating each plot for each dimension (b). I address the focus point selection problem by sampling over the parameter space and then projecting the slices in the corresponding plot (c). The user can mouse over a particular slice in one plot and the corresponding slices are highlighted in the other dimension plots (d). This allows one to see the corresponding function behaviors in the other dimensions. Finally, one can cluster the function slices (e), to show groups of similar behavior in the manifold.

2.1 Motivation

The visual analysis of multi-dimensional scalar functions is a fundamental aspect in many areas, from computational sciences (based on computer simulations) to data sciences (based on machine learning techniques) and general optimization algorithms. Neural networks, for instance, have shown to produce very good results that come in the form of highly non-linear response manifolds. While understanding these manifolds would greatly help to verify the resulting models, there is currently no established way to inspect these multi-dimensional manifolds. In case of an optimization algorithm, a good visualization of response manifolds could, for instance, help to see where possible “holes” (local extrema) are that the algorithm may get trapped in and fail to find the global optimum. By visually examining the optimization functions, one can develop new insights on how to improve the optimization algorithm. This work is based on the assumption that visual analyses of these functions will help to increase understanding of what they are doing, as argued by Gleicher [38].

To examine these functions visually they must be reduced to be displayed on a 2D screen. Currently, the two major approaches used to visualize these spaces are either two-dimensional slices, a technique known as HyperSlice [119] or dimension transformation techniques. This includes dimensionality reduction and topological methods [24, 21]. HyperSlice shows two-dimensional slices (using either a heatmap or contour view) of the function directly around a particular focus point. It clearly shows the behavior of the function with respect to the parameters. However, one can only view one focus point at a time and the approach does not scale well to many dimensions. With each additional dimension one must substantially shrink the subplots so less detail can be seen just like in scatterplot matrices. Topological and dimensionality reduction techniques take the opposite approach. They morph the space to produce a 2D global view. However, the morphing process is rather complex and it is unclear what the resulting layout means. This reduces comprehension. Ideally we would like to somehow combine the global view with local detail.

In this chapter, I explore the idea of 1D slices to fill this gap. I focus on multi-dimensional continuous scalar functions. I define these as functions that take two or more scalar parameters as input and produce a single scalar as output. Towards illustrating the benefits, I propose a concrete technique using projections of 1D slices, called Sliceplorer [113] (Figure 2.1). 1D slices are traditional function line plots familiar to anyone with basic mathematics knowledge: one dimensional curves with respect to changes in a single parameter. Like HyperSlice, they show a separate subplot for each input dimension. For 1D slices, the number of subplots scales linearly with the number of dimensions, not quadratically as with 2D slices. I address the issue of having to choose a focus point by sampling focus points and then showing all slices as a projection. Therefore, my 1D approach can be seen as a hybrid method of slicing and projection techniques.

In order to evaluate any new technique, we need to consider what data characteristics and tasks each method is good for. As of yet, there has not been a comprehensive listing of the tasks a user would want to perform when looking at multi-dimensional continuous functions. To this end, I begin development of this task summary by extending the task classification of Amar, Eagan, and Stasko [2]. I use this classification to evaluate Sliceplorer and compare it to 2D slices and different topological approaches. This comparison characterizes which technique is best for which tasks and reveals that 1D slices is the most flexible of the current approaches. That is, it supports the broadest range of different tasks.

I also provide three usage scenarios comparing Sliceplorer with other state of the art techniques. These scenarios illustrate that 1D slices can reveal structure in the functions that could not have been seen before. For example, I discuss how one can use my method to compare the *global* prediction manifold of a neural network algorithm against a support vector machine to guide and better understand the architecture of a neural network. This is currently an open research question in the machine learning community.

In summary my contributions are:

- exploring the idea of 1D slices with a concrete software prototype called Slice-plorer,
- a task-based analysis of multi-dimensional functions, comparing 1D slices to 2D HyperSlices and topological techniques, and
- three usage scenarios illustrating the value of 1D slices in common function visualization scenarios.

2.2 Related Work

The question of how to comprehend multi-dimensional data is a heavily researched area in visualization. There are two principal approaches, projection techniques (such as scatterplots) and slicing techniques (such as HyperSlice). Projection techniques generally show all of the data and, therefore, represent a more global view. On the other hand, slicing techniques present more of a local view around a point of interest (which we call focus point). The lion's share of previous work is concerned with projection approaches of discrete data. In contrast, I focus on continuous multi-dimensional scalar functions and seek to combine the strengths of projection and slicing. Given the focus on continuous data, the options for a visual exploration are limited and could be categorized into three areas (a) discretization, (b) local methods, (c) global methods.

2.2.1 Discretization

There are a number of different approaches to display discretized multi-dimensional functions. The typical approaches are scatterplot matrices [46], parallel coordinates [58], star coordinates [62], and RadVis [51]. Star coordinates and RadVis were generalized into one framework by Lehmann and Theisel [69]. These can all be combined with a variety of dimensionality reduction techniques [52]. However, all of these seem inappropriate if the mental model of the function we are studying is a continuous one. In such a case all of these projection techniques would fail to properly convey the complexity of the underlying continuous phenomenon. Hence, while discretization

seems like an easy way out, it is not a proper alternative for studying *continuous* multi-dimensional functions, such as regression functions or classification boundaries. Here, one of the main concerns is understanding a continuous phenomenon and a good visualization design should thus respect this mental model [116, 99, 74].

2.2.2 Local methods

The idea of a local technique is to focus on a part of the function. Interaction is used to explore other parts of the function. One of the oldest approaches here is the HyperSlice technique by van Wijk and van Liere [119]. HyperSlice is a technique where the function is shown directly but in multiple 2D slices laid out similar to a scatterplot matrix. In many ways, Sliceplorer was inspired by this work. One of the drawbacks of HyperSlice is that one has to choose a focus point — a point common to all 2D slices. Exploring the full data set then shifts over to exploring all possible focus points. Although not created for HyperSlice specifically, techniques like the grand tour [5], projection pursuit [55], and optimal sets of projections [68] might be appropriate to tackle this issue. All of these approaches are still local though. A mental image of the global function can only be built up over time and with mental effort by browsing through the focus points. Our approach seeks to overcome this limitation while keeping the benefits of ease of understanding.

Note that for some tasks a local view might in fact suffice, such as when one is interested in the robustness of an extrema value. For example, Tuner [112] used 2D continuous slices, letting the user navigate them via selecting Pareto optimal focus points in a separate view. Berger et al. [10] use coordinated views of scatterplots and parallel coordinates to show additional (continuous) prediction uncertainty. Having a discrete approach provided extra space to display information about the prediction uncertainty for the currently selected point.

2.2.3 Global methods

The visualization community has developed many global views of multi-dimensional continuous functions. Continuous scatterplots [6] and continuous parallel coordinates [49] can encode a multi-dimensional density field into either two dimensions or more than two dimensions respectively. Here, we are not as concerned with the density field, rather we are concerned with the manifold created from a continuous multi-dimensional scalar function.

Topological methods like topological spines [24], the work by Gerber et al. [36], and contour trees [21] extract extrema and saddle points from a function and then show these. These methods are good for seeing the relative relation of extrema in a function. However, they do not work for important tasks like robust optimization. Here, one does not necessarily want to find the global optimum but wants an optimum in a relatively “flat” area of the parameter space.

With Sliceplorer, we use line plots together with the widely-used technique of projection to overdraw 1D slices. This approach is similar to the work by Hall et al. [42] but differs in two major ways. (1) They showed 2 primary dimensions with slices and then used the third for color limiting their view to three dimensions and (2) they were concerned with isosurface extraction. This technique can scale to any number of dimensions and the evaluation is based on a much broader set of tasks and applications, such as parameter space analysis.

2.3 Sliceplorer

When developing Sliceplorer, I first identified design requirements with respect to tasks that a user would perform when analyzing multi-dimensional scalar functions. I continuously evaluated how the technique fits with these requirements and iteratively adapted it to encompass as many tasks as possible. A static slice view itself does not address many of the tasks required so I use interaction methods to address these and create a comprehensive technique.

2.3.1 Design requirements

When analyzing a function visually, there are a number of features that the user wants to see.

R-peaks: The most obvious feature is identifying peaks and valleys. This is primarily done to find the global optimum of a function.

R-robust: The relative height around each optima is important to detecting the robustness of that optimum. In some cases one may prefer a local optimum over a global one if it is more stable. This is very common in simulations of manufacturing processes where variations in manufacturing tolerances should not affect the performance of a part too much [10].

R-bowl: Similarly, we may be interested in how “bowl”-shaped the area around an optimum is. The goal is similar to robust optimization but rather than looking for “flat” areas of the function we are looking for areas with smooth gradients. The amount of this smoothness is important for correctly parameterizing optimization algorithms [7]. An incorrect parameterization can either cause them to get “stuck” at some local minimum or make unreasonably slow progress towards the global minimum.

R-overall: Finally, we want to understand the overall “shape” of the function. It is important to understand if it is smooth everywhere and how much variance there is in the function. When building a surrogate regression model we need to know if the function has consistent variance and we need to choose a model that captures this behavior.

All of these requirements mean that we need to view more than just the maxima and minima of a function.

2.3.2 Intuition

If we were analyzing one-dimensional continuous functions then the choice would be obvious: a line plot like the one in Figure 2.1a. The x-axis is used for the independent variable or parameter setting and the y-axis is used for the dependent variable or scalar

value of the function. The function response is shown with a line. This is a metaphor that anyone who has taken high-school algebra can comprehend. The vertical and horizontal location channels visually encode the primary values of interest. These are the “best” visual channels to use in terms of accuracy and sensitivity to differences according to Bertin and others [12, 76].

Figure 2.1 shows the evolution of our technique from a one-dimensional function to the full, multi-dimensional Sliceplorer view. To extend this simple technique to multi-dimensional functions, each parameter receives its own one-dimensional plot (Figure 2.1b). There will be d plots where d is the number of dimensions in the function. In the same way that HyperSlice [119] is inspired by the SPloM layout, one can use any layout technique for multiple histograms. 2D slices scale as $\mathcal{O}(d^2)$ which is worse than the $\mathcal{O}(d)$ for 1D slices. 1D slices also give us separable channels remaining for encoding of additional information such as uncertainty or optimization traces (see Sec. 2.5.3).

Slicing offers a number of advantages over projection-based views like scatterplots and histograms. Slices give context around a particular focus point. We can see the precise shape of the function at this point. For example, peaks and valleys (**R-peaks**), flat areas (**R-robust**), and variance in the function can all be seen directly. While scatterplots and histograms can be used to see general trends, they suffer from “false distances” where points that are visibly close to each other are not actually close to each other [124].

2.3.3 Focus point projection

Showing a local 1D slice requires the selection of a single focus point, i.e. a point in multi-D through where all 1D slices intersect. Once this focus point is selected, one can use an off-the-shelf 1D function plot drawing method to draw the slice itself. Rather than only showing one focus point (i.e. one slice line per dimension) at a time and having the user choose a focus point Sliceplorer selects multiple focus points automatically. This enables a more global view of the function (**R-overall**). Now, all 1D

slices (one per focus point and dimension) are projected onto the same plot (see Figure 2.1c). In doing so, users do not need to memorize the previously seen slices, they can look among them to see general trends. This approach combines the ideas of slicing and projection, and fosters one of the core strengths of visualization: “perception beats recall” [84].

I use a Sobol sequence [107] to select the focus points themselves. The Sobol sequence is a space-filling, quasi-random, low-discrepancy sequence that is designed for sampling in high dimensional spaces. The Sobol sequence will sample the multi-dimensional parameter space with an economy of samples. This will maximize the chances that we will see extrema (**R-peaks**), plateaus (**R-robust**), and bowls (**R-bowl**) in the 1D slices. In addition, using a Sobol sequence makes it easy to adjust the number of 1D slices shown (i.e. focus points) on the fly. Specifically, it avoids a complete resampling of the parameter space like we would need with a Latin hypercube sampling.

2.3.4 Linked selection

One disadvantage with 1D views over 2D views is that we cannot see the two-dimensional interactions anymore. I compensate for this with interaction. The user can mouse over a particular slice which will highlight all slices corresponding to that focus point. That is, one slice in each view is highlighted. I also superimpose the focus point itself on these lines. In doing so, the user can see the behavior of the function with respect to the other parameters around that focus point (see Figure 2.1d).

2.3.5 Clustering

Similar to visual encoding techniques such as parallel coordinate plots, projected 1D slices might mask certain patterns due to overdrawing. Figure 2.2a is an example where it is difficult to tell if the slices are monotonic or bowl-shaped (**R-bowl**). The interactive slice highlighting can give some insight into how individual slices are behaving but lacks a global method to distinguish groups. Sliceplorer offers a clustered

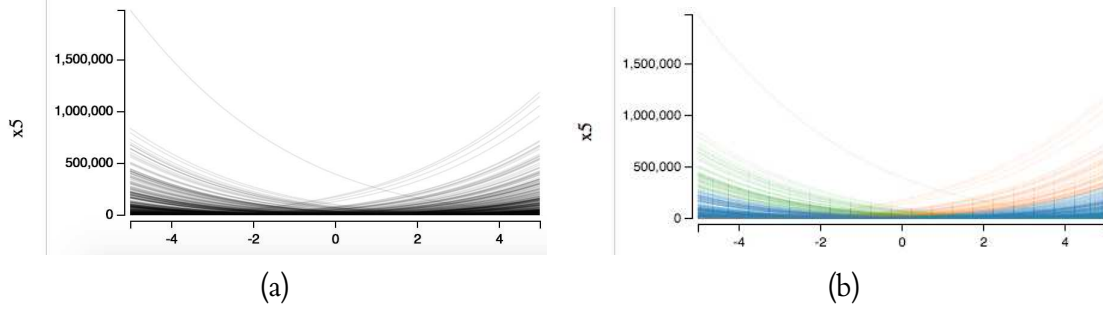






Figure 2.2: 500 projected slices of the 5th dimension of the 5D Zakharov [7] function. It is difficult to see if the slices are bowl shaped or two sets of monotonically decreasing and increasing curves. It is much clearer in the cluster view that there are actually three sets of curves: there is a set of monotonically decreasing curves and a set of monotonically increasing curves. The very low-value curves form a third set.

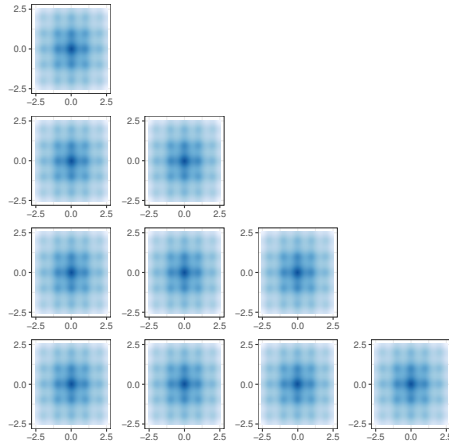
Table 2.1: This table shows the summary of the task-based evaluation. I extended the discrete data-focused tasks of Amar, Eagan, and Stasko [2] to directly address continuous data, with the exception of “sort” (see section 2.4). The table shows the scores from our qualitative results inspection as well as the expert study on a scale from “none” , “partially” , “mostly” , to “fully”  where “none” means that the task is not addressed at all and “fully” means that this task is directly supported by this view. There are quotes of the general description from Amar, Eagan, and Stasko’s paper in the “discrete” column for reference.

Task	Task description for discrete data items from [AES05]	Our adaption to continuous scalar functions								
			QRI results				Expert study results			
			Hyper-slices	Topological spaces	Contour tree	Gruber et al.	Hyper-slice	Topological spaces	Contour tree	Gruber et al.
Retrieve value	"Given a set of specific cases, find attributes of those cases"	Given an x , what is the function value?								
Filter	"Given some concrete conditions attribute values, find data cases satisfying those conditions."	For what parameter values is the function equal or over x ?								
Compute derived value	"Given a set of data cases, compute an aggregate numeric representation of those data cases"	Summary statistics: variance, mean, SA								
Find extremum	"Find data cases possessing an extreme value of an attribute over its range within the data set"	Find local/global min/max								
Determine range	"Given a set of data cases and an attribute of interest, find the span of values within the set"	What is the range of possible outputs?								
Characterize distribution	"Given a set of data cases and a quantitative attribute of interest, characterize the distribution of that attribute's values over the set"	What types of shapes do the manifolds have								
Find anomalies	"Identify any anomalies within a given set of data cases with respect to a given relationship or expectation, e.g. statistical outliers"	Do areas of the manifold have shapes unlike any others								
Cluster	"Given a set of data cases, find clusters of similar attribute values"	Areas of the manifold have similar shapes								
Correlate	"Given a set of data cases and two attributes, determine useful relationships between the values of those attributes"	1D vs 2D relationships								

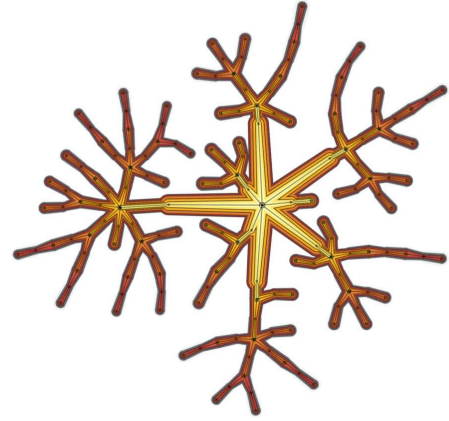
slice view to address this (Figure 2.2b). The clustering is done with a k -nearest neighbor algorithm using the L^2 distance between two slices as the distance metric. This allows us to group the slices into distinct groups of behavior and color-code these groups to distinguish them.

2.4 Task-based evaluation

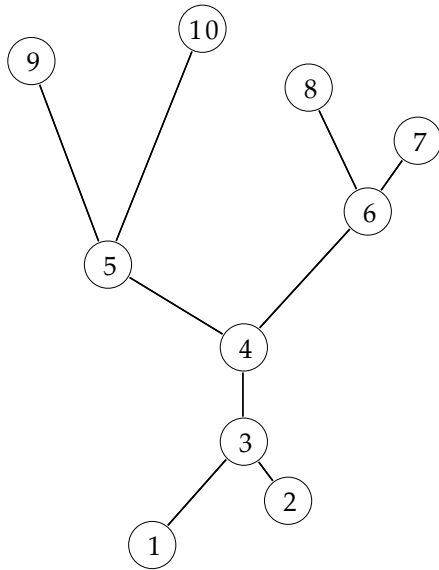
I first evaluate 1D slices in terms of their flexibility to deal with a broad set of different low-level tasks. Task taxonomies give a basis for comparing visualization techniques



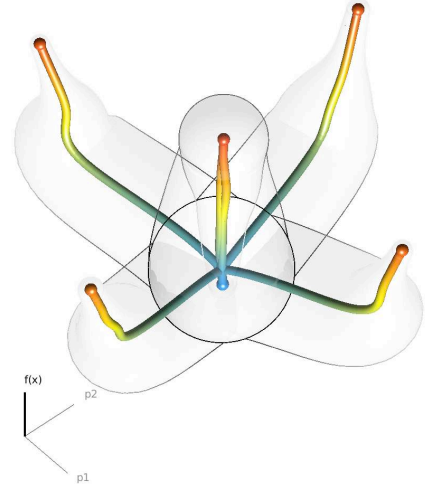
(a) HyperSlice [119]



(b) Topological spine [24]



(c) Contour tree [20]



(d) Gerber et al. [36]

Figure 2.3: The four techniques we used to compare with 1D slices. With the exception of HyperSlice, the images are from the respective papers and show different datasets used in their context.

to each other [84]. If a technique addresses a large number of tasks, that is usually a good indicator of its flexibility. Over the last years, many different taxonomies have been proposed [2, 16, 48, 98]. However, to the best of my knowledge, none of these taxonomies thus far had a dedicated focus on the visual analysis of multi-dimensional *continuous* data. I thus took a popular taxonomy for tasks on discrete data, by Amar, Eagan, and Stasko [2], and extended each of their task categories to directly address continuous data. This is an initial step towards more consideration of multi-dimensional continuous data as a first class citizen when developing task hierarchies.

Using this list of tasks, I compare 1D slices to other state of the art techniques for

multi-dimensional continuous data: HyperSlice [119], topological spines [24], contour trees [20], and the technique by Gerber et al. [36] (see Figure 2.3). We refer to topological spines, contour trees, and the work by Gerber et al. as topological techniques when it makes sense to compare them as a group. We evaluate based on all tasks, except for “sort”, for which I could not find a suitable extension to continuous functions. The guiding theme in the extensions is that users want to view the relationship of independent variables to the dependent variable and to see how the dependent variable changes with respect to the independent values. The extensions are shown in Table 2.1 along with the results of two investigations we conducted based on them, as detailed in the following section.

2.4.1 Study design

To perform a task-based evaluation, I investigated the different techniques in two different ways. First, I used a *qualitative result inspection* approach [59]. I iteratively analyzed the techniques with different datasets and summarized our discussion and analysis on a four point scale: “None” means that it is not possible to perform the task with the technique, “partly” means that it requires major interaction with the view to accomplish the task, “mostly” means that one can accomplish the task with little interaction, and “fully” means that this task is directly addressed by the technique.

Second, in order to get a more objective judgment I also asked *four visualization experts* familiar with examining multi-dimensional spaces like parameter space exploration to examine the eight datasets with different techniques and rate how well each task can be accomplished with each technique on the same scale. Figure 2.3 shows the averaged results along with the results of the qualitative result inspection in Figure 2.3.

For the techniques, I use my own implementation of HyperSlice and topological spines since no code was available. I used the `msr` R package [37] which implements the algorithm of Gerber et al. [36]. For the contour tree I used the `libtourtre` library and then rendered the trees using GraphViz using the Sugiyama [35] layout. As datasets, I chose the 2D sinc function, 5D Rosenbrock [95] function, 6D Ackley function, a 26

node hidden layer neural network built on the Boston housing dataset [70], a support vector machine with Gaussian kernel built on the housing dataset, the fuel 3D volume dataset [94], and the neghip 3D volume dataset [94]. Not all datasets could be rendered with all techniques due to software errors.

2.4.2 Results

In this section, I summarize our discussion about the strengths and weaknesses of each technique in terms of performing the task. For more details, there is also a website that contains details of how each visualization technique can solve each task. The website is available at <http://sliceplorer.cs.univie.ac.at>.

Retrieve value: In the discrete case, the user should be able to look at a point and get the detailed values of it. In the continuous case we are interested in what the function value is for a certain input parameter setting. All the techniques support this although with the topological methods this is only possible for the extrema and saddle points as all other points are filtered out. For example, there could be many points between node 4 and 5 in the contour tree (Figure 2.3c). With slicing techniques, both 1D and 2D, the values can be read directly off the chart. Of course, for all techniques the adding of interaction, such as a tooltip, can make retrieving concrete values even easier.

Filter: Amar, Eagan, and Stasko describe this task as a general filtering query on data points. In the continuous case, the user wants to understand the outputs of the function. This is a query as to where the function value is in a certain range. With continuous data this is the domain of isosurface extraction. This is possible with slicing techniques by visual examination. With HyperSlice, though, one must be careful to view sufficient focus points to get a general idea of where the function equals certain values. Topological spines also shows this directly and they use concentric areas (Figure 2.3b) to give a general idea of the area that a particular value range takes up. The other topological techniques allow one to see if a certain value is possible, for example, we can see that the function represented by the contour tree in Figure 2.3c

takes the values greater than 4 somewhere by seeing that there are edges from node 4 to nodes 5 and 6. However, there is no relation back to the parameter settings that will produce these values.

Compute derived value: The direct interpretation of this task to continuous data is to compute derived value results about the curves like mean and variance. Many of these values can also be perceived visually. Topological methods compute the persistence value between the function to determine what to show but with the exception of topological spines this is hidden from the user. Topological spines show a graph of the persistence and “saturated persistence” which allows the user to select which nodes to filter. Projections of 1D curves allows us to see the distribution. In Figure 2.1c we can see that there are very few function values around the global minimum and the function has two types of behavior: a periodic sine wave across the domain and a general parabola shape.

Find extremum: All the topological techniques we evaluated support this in some way. With HyperSlice one needs substantial guidance on setting the focus point to find extrema (like a histogram of function outputs). 1D slices is a global technique showing all slices at once, one can find extrema by inspecting the graphs. As previously mentioned, topological methods are purpose built to extract extrema from continuous data. For example, it is easy to see that the function using the method by Gerber et al. (Figure 2.3d) has five maxima and the function of the contour tree (Figure 2.3c) has four maxima.

Determine range: Amar, Eagan, and Stasko describe this as finding the range of possible values for a particular attribute. There is really only one attribute of interest: the values of the multi-dimensional scalar function. Any view from which we can read the global minimum or the global maximum allows us to do this. Contour trees, Gerber et al., and 1D slices all allow us to read these off the view. Topological spines either show the global maximum *or* the global minimum, but not both. HyperSlice has no way to do this directly by adjusting the focus point. However, one expert noticed that they could simply read the range of the function off of the color legend.

Characterize Distribution: Here again there is one key value of interest that we

want to characterize: the function value. This requires a global view. Projections of slices directly show how the function slices are distributed. We can see in Figure 2.1d that there are very few function values around the global minimum but many around high values. It would be difficult to use HyperSlice to truly understand the distribution of values. The user would somehow have to browse around the focus points and then memorize the function values. Topology throws away the spatial element and just shows the relationships between extrema and saddles.

Find anomalies: Anomalies in the discrete case are single point outliers. While that is also possible in the continuous case, we may also have entire parts of the function that are unlike any other part. These should also be identified. In a global view like projected 1D slices these will show up visually. The slices will stand out from the rest similar to other projection-based techniques like scatterplots. With HyperSlice we must browse around until we can see one directly. However, we will see it if we can find it. Topological methods will only show extrema in terms of maxima or minima values but not shape and hence mask anomalies and outliers.

Cluster: Since we are looking at manifold behaviors, we want to be able to group the functions into areas of similar behavior. For example, are they monotonically increasing or decreasing? Furthermore, can we find areas where the variance changes? The topological technique of Gerber et al. [36] was created to address just this. They split the function into areas of monotonic behavior and then show a line indicating how those monotonic regions are related to each other. However, the way they reconstruct the function between extrema and saddle points does not allow us to view the variance between these points as the 1D slice view allows. Clustering the 1D slices tries to split the slices into groups of similar behavior (see Figure 2.2b).

Correlate: Finally, we consider correlation. In the discrete data case the goal is to find correlation between attributes. With continuous data, we already have a dependency between the independent and dependent variables. What we would like to learn is how many variables have an influence on the function. With 2D views (that only HyperSlice provides) one can see both 1D and 2D interactions with the function. We can see that the function in Figure 2.3a has radial behavior so the function value

depends on both 1D and 2D interactions. None of the other techniques are capable of showing 2D interactions between parameters.

2.4.3 Summary

From the summary in Table 2.1 we can see that the 1D slices technique addresses more of the tasks than any other technique. It is not always the highest performing view though. HyperSlice is the only technique evaluated that could show more than one-dimensional interactions but it does not do well on global tasks like extrema detection. The various topological techniques directly address tasks related to extrema detection and comparison but do not perform as well on others. The experts often commented that they felt they needed more knowledge about what exactly the topological techniques were doing in order to interpret the results. Thus, the ratings for these techniques may be artificially low. I conclude that 1D slices are a very flexible technique indeed.

2.5 Usage scenarios

In addition to evaluating 1D slices with a low-level task hierarchy, I also provide usage scenarios to understand their value in real-world applications. First, I begin with an illustrative example using the 2D sinc function. I then use our 1D slices approach to illustrate how it can help better understand neural network architectures for a regression problem. Finally, I use 1D slices to investigate the effect of initial position on optimization algorithm performance.

The purpose of these evaluations is a proof of concept that 1D slices can be used for real-world problems. In particular, it is not meant as a comparative evaluation as provided in the previous section. To the best of my knowledge, neither HyperSlices nor topological techniques have been applied to understanding neural networks nor optimization algorithms so far. A full adaption of, and comparison to, these techniques for the provided use cases are beyond the scope of this thesis, and are left for future work.

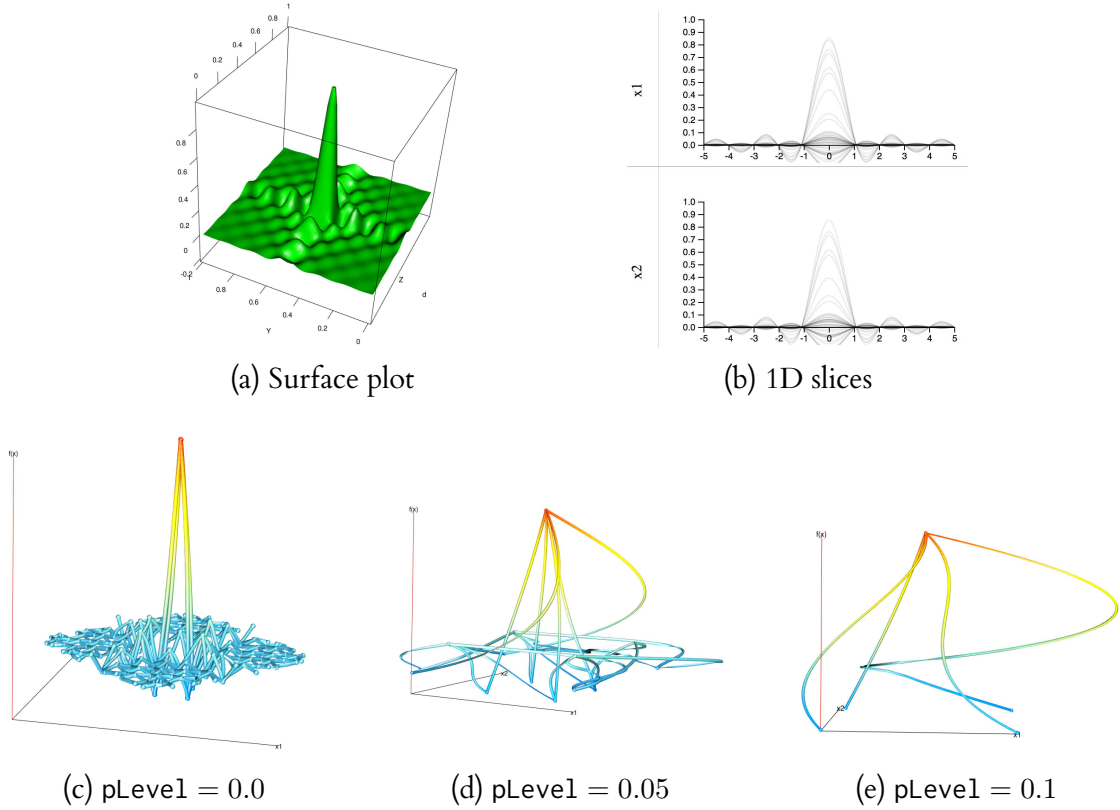


Figure 2.4: Different views of the 2D sinc function. I show the surface plot in (a) for reference. My 1D slice view is shown in (b). The central peak as well as the sub peaks are prominent. For comparison I show the method of Gerber et al. [36] in (c)–(e) at different levels of persistence filtering. With no filtering (c) the graph looks much like the original function. The plot is very sensitive to the filtering level. (d) and (e) are all very different from each other.

2.5.1 2D sinc function

Imagining how a function in more than 3-dimensions looks is difficult if not impossible. In order to illustrate how Sliceplorer visualizes functions we show the 2D sinc function. I am using the formulation where $y(x_1, x_2) = \frac{\sin(\pi x_1)}{\pi x_1} \frac{\sin(\pi x_2)}{\pi x_2}$. For reference, Figure 2.4a shows a 3D surface plot of this function. The global maximum is at $x_1, x_2 = 0, 0$ where $y = 1$. There are a number of local maxima and minima of decreasing value radiating out from the origin.

Figure 2.4b shows the 1D slice view using Sliceplorer with 50 slices in each of the 2 plots. We can clearly see that the maximum value occurs when $x_1, x_2 = 0, 0$ in the graph at around $y = 1$. We can also see the decreasing extrema radiating out from the origin. In addition, we can precisely measure the height and x-location of the

extrema. If we want to examine a particular trace then we can highlight it in the view and see the full slice highlighted on screen.

For comparison, I show several visualizations of the 2D sinc function rendered using the `msr` package [37] in R. This package implements the visualization of the Morse-Smale complex from Gerber et al. [36]. I sampled the function with 2000 sample points using a Sobol sequence. The 1D slices view is showing 50 focus points with 21 samples for each slice. I chose 2000 sample points in order to use a similar sampling method and number of samples to the Sliceplorer method. The function can do persistence-based filtering of the graph before rendering. This is controlled by the `pLevel` parameter which filters all persistences less than a certain value. In Figure 2.4c we show the view with the filtering level set to 0, i.e. no filtering. The view does a very good job showing the critical points of the graph. It looks very similar to the surface plot (Figure 2.4a). However, the visualization is very sensitive to the filtering level. In Figure 2.4d and Figure 2.4e we show the sinc function with the filtering level set to 0.05 and 0.1 respectively. The 1D slice view does not suffer from this issue of parameterization.

2.5.2 Neural networks

Artificial neural networks are currently gaining a lot of attention in machine learning. The goal of these algorithms is to produce a multi-dimensional function fitted to the training points. Neural networks, in particular, have proven to be very good at producing accurate, generalizable predictions. One of the major challenges for designers of such models, however, is to properly architect these networks. For instance, how many hidden layers does one need and how many nodes should be put into each layer? These architectural choices can drastically change the predictions. While these choices are crucial, currently, there is only little guidance available for designers. A typical rule of thumb is to use a hidden layer two times the size of the input dimensions. There are also some general proofs regarding what type of functions neural networks can represent [54, 32]. However, there are no formal guidelines

Neural network – 26 SVM – polynomial Neural network 5+3 SVM – radial

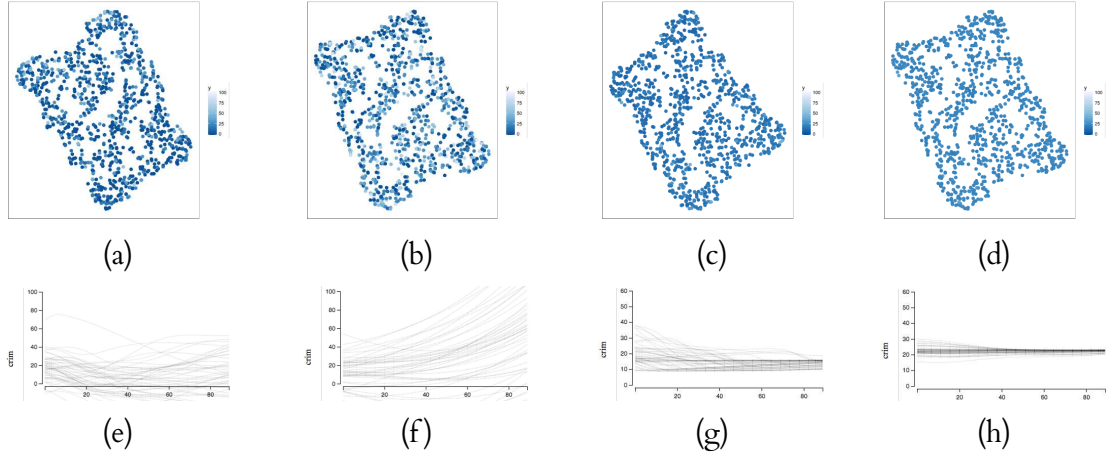


Figure 2.5: Two different views of the predictions of four different machine learning regression models on the Boston housing dataset. The top row (a – d) shows predictions by each model compressed to two dimensions with t-SNE. The point color indicates the function value on a continuous color scale with dark blue being 0 and light blue the highest value. The bottom row (e – h) shows a 1D slice view of the first dimension of the dataset, crime rate. We show the slices of the remaining dimensions in chapter A. The 1D slices reveal interesting information about how the models perform and can assist with model selection. We may not want to use the SVM with polynomial kernel (f), for example, since it predicts that home price will go up with higher crime rates.

for designing these networks [39] and the way these models make predictions is still obtuse.

One of the ways we can increase the understandability of neural network regression models is by viewing the response function directly [38]. If we want to understand how the network architecture affects the prediction we could compare the prediction manifold to one produced by a “simpler” machine learning model [93], for example a support vector machine [105]. Support vector machines have known guarantees on error rate with the number of training samples. With this comparison we may be able to get some better insight about how the neural network learning algorithms are performing.

To compare, I chose the Boston housing dataset [70] from the UCI repository. This dataset contains median home prices given 13 factors including crime rate, age of the house, and proximity to highways. I then trained a neural network with a single hidden layer of 26 nodes, a neural network with 2 hidden layers: one of 5 and one

of 3 nodes, a support vector machine with a polynomial kernel, and a support vector machine with a radial (Gaussian) kernel.

I compare 1D slices with an adaption of the common way of viewing classification algorithms to continuous data. The results of *classification* models are commonly visualized by using MDS [66] or t-SNE [118] to reduce the input dimensions to two and then present a scatterplot with the predictions colored by class. I extended this by sampling the prediction model with 1,000 samples from a Latin hypercube [110] (a space-filling design), converting the points to two dimensions with t-SNE, and then coloring the points on a continuous scale which shown in Figure 2.5 (top row). The bottom row of the figure shows the 1D slice view of the same four prediction models. Only the first dimension is shown due to space reasons. The full 13 dimension slice view image in chapter A.

Showing the changes in home price as it corresponds directly to the crime rate can help to increase confidence in a model. From the prediction lines, one may not want to use the SVM with a polynomial kernel. By and large the prediction lines are increasing. This means that the home price is increasing as the crime rate goes up. This does not really make sense. The model is not generalizing well. Similarly, the neural network with a single hidden layer (left column) also has a number of curves that increase as crime increases. The neural network with two hidden layers does not have this problem. Maybe this is the best model to use in this case.

In summary, this usage scenario illustrated that a direct inspection of a model's response surfaces can give intuition of its behavior, and can lead to a better model selection and a better intuition of the modeling process. 1D slices can help to gain important insights in this process.

2.5.3 Optimization algorithm

General purpose optimization algorithms try to find the global minimum (or equivalently, the global maximum) of a function of arbitrary dimension. Many optimization algorithms such as Nelder-Mead [86] work by starting at a particular parameter set-

ting and evaluating the “shape” of the function around that point. The algorithm then determines where the function is decreasing the greatest, and “jumps” a certain distance in that direction. The “jump,” starting position, and termination tolerance parameters are user-settable parameters. Depending on how they are set, the algorithm can get stuck in a local minima or take unreasonably long to finish.

When one is trying to parameterize or build optimization algorithms then one wants to evaluate the trace of the optimization on an easy function that is fast to compute first. This analysis helps to better understand how to parameterize for more complex problems but are too computationally expensive to analyze directly. Visual inspection of the easy function before running the optimization algorithm, as well as viewing the trace of the optimization algorithm (the sequence of steps it took) is a good way to ensure that the algorithm is converging towards the global minimum.

I compare 1D slices with HyperSlice [119] as this is the only technique that also directly visualizes the parameter space. I ran the Nelder-Mead optimization algorithm on the 5D Ackley function [1], a popular optimization algorithm testing function. To examine the effect of starting position, I tried different starting positions: $x = (1, 1, 1, 1, 1)$ and $x = (2, 2, 2, 2, 2)$. Figure 2.6 shows the trace of the optimization algorithm overlaid on both 1D slices and HyperSlice.

The 1D slice view allows us to see the path that the algorithm took and the general shape of the function simultaneously. In addition, the 1D slice view shows that the distribution of values around the global minimum is small. Most of the slices are clustered around $y = 10$ with only one slice descending close to $y = 0$. Since the sampling is uniform in the parameter space this means that it is very difficult to select slices around the global minimum. In fact, this is a known property of the Ackley function. It is easy to see that the optimization algorithm got stuck at a local minimum when started at $x = (2, 2, 2, 2, 2)$. However, with the HyperSlice view it is difficult to see the difference in value and steepness of the function at $x = (1, 1, 1, 1, 1)$ versus $x = (2, 2, 2, 2, 2)$. Humans are not good at perceiving fine differences in color [84], but is required for this task. We learn a lot more about the behavior of the optimization algorithm from the 1D slice views (see Figure 2.6a and Figure 2.6c) than the

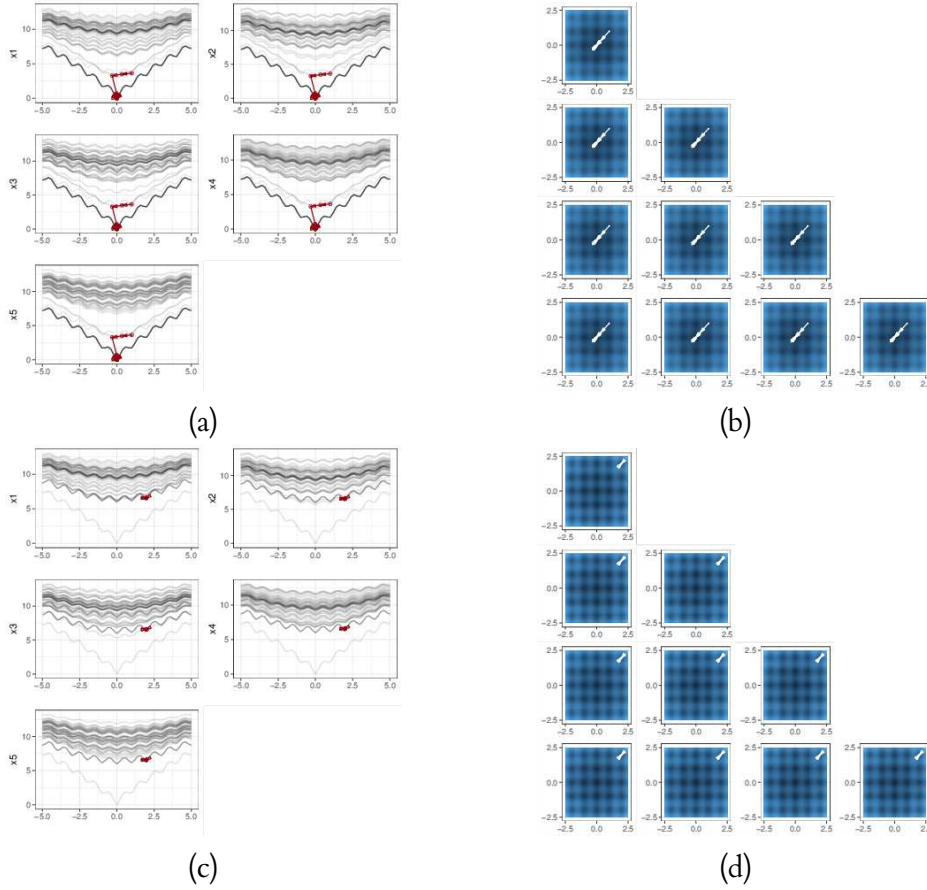


Figure 2.6: 1D slice and HyperSlice views showing traces of an optimization algorithm searching for the global minimum of a 5D Ackley function. (a) and (b) show the trace starting at the point $(1, 1, 1, 1, 1)$ while (c) and (d) show the trace starting at the point $(2, 2, 2, 2, 2)$.

HyperSlice view. However, the HyperSlice view does clearly show that that optimization algorithm is moving in multiple directions at once. This is not clear in the 1D slice views.

2.6 Discussion

The above examples illustrated that the technique of 1D slices as presented are quite flexible and useful for various low- and high-level tasks. However, I do not intend to claim that it is the only and best method for all problems out there. Rather, I would like to argue that it is a valuable (and thus far overlooked) technique in a toolbox of visual inspection methods for multi-dimensional functions. I hope that this work inspires a discussion and exploration of guidelines for tasks, proper visual encoding,

and interaction techniques for various application areas. Along these lines I would like to put forth our current experience with various techniques.

Topological techniques are helpful for a global overview: Topological techniques allow us to compare *between* optima but are not as good at evaluating the area *around* an optimum since these areas are typically abstracted away. Topological spines attempts to compensate for this by showing the area covered by a particular optimum as an area around the node. However, many of the tasks like “correlate” and “cluster” are best served by viewing the response manifold directly. In a larger system, the topological techniques could be used effectively as a global overview of the function with a HyperSlice or 1D slice showing local context. Selecting a point in the topological view would change the focus point in the local view.

HyperSlice is good when you need to show 2D interactions: HyperSlice is the only technique that can display more than one dimension of data interaction. So, if this is a requirement then HyperSlice is the best option. However, one can use 1D slices to get a general overview of the dependence of the function on each dimension. The dimensions that are not interesting because, for example, the function is not sensitive to them could easily be eliminated from further consideration. This would reduce the number of subplots that we need to view in the HyperSlice plot.

1D slices should be used for a “first pass” visualization: 1D slices addresses many of the tasks that a user wants to perform. The technique does a very good job on a wide variety of tasks. 1D slices are easy to implement, easy to understand, and the static view provides a lot of information.

2.7 Limitations and future work

The 1D slice view consists of a projection of many lines. The distribution of slices are shown through direct projection. Techniques like contour boxplots [123] and curve boxplots [78] build a distribution model of curves which could help to address the “characterize distribution” task in Table 2.1. However, neither of these or any of the other time curve visualization techniques have been applied to multi-dimensional

functions. Evaluating these techniques for this purpose is an exciting topic for future work.

When developing the 1D continuous slicing technique I only considered multi-dimensional continuous scalar functions in terms of requirements, tasks, and comparisons. I do not consider multi-field (i.e. functions with multiple outputs) or complex-valued functions in the analysis. There are multi-field topology techniques to address this [30, 56, 19] which I do not consider but the technique and analysis would need to be extended to this domain. This is left for future work.

The x-axis of each 1D slice is independent of the x-axes of the other 1D slices. This allows each plot to scale individually if the range of inputs have different values. The x-axis and y-axis automatically change to incorporate their respective minimum and maximum ranges. While the x-axis scales itself independently, the y-axis is the same for each plot. This is also the default behavior in many of-the-shelf plotting packages. The plots will adjust automatically to shifts. For the x-axis we use axis-aligned projections. Therefore, the views are sensitive to rotational transformations of the function.

Finally, the Sliceplorer technique is also based on sampling, just like the techniques used in the comparison. As with any technique based on sampling one must be careful to take an adequate number of samples in order to properly capture all desired behavior. If the function is not smooth we may see a slice that is an “outlier,” i.e. one slice is much higher or lower than all the others. In this case all other slices will be compressed into either the top or bottom of the chart. This is often a problem with many common visualization techniques like bar graphs or scatterplots and can be addressed with log scaled axes, for example.

2.8 Conclusion

In this paper we have presented Sliceplorer, a visualization method for multi-dimensional functions based on one-dimensional slices. We defined a task taxonomy specific to multi-dimensional continuous functions and found that, while some state of the art

techniques are very good at addressing specific tasks, our method supports a wide variety of tasks. Consequently, our technique may be a good first pass when visualizing multi-dimensional continuous functions. It is easy to implement, easy to understand, and addresses a greater variety of tasks than any other technique.

3 | 2D slices

One-dimensional slices cannot do everything. When we are examining shapes then we need to examine the relationship between dimensions. Thus, 2D slices are the minimum number necessary to examine these relationships (see Table 2.1). If we have a function describing the multi-dimensional shape we are examining then we can generate two-dimensional slices by constraining all but two of the parameters to a particular value and then computing the two-dimensional outline directly. However, there are domains, such as polytopes, where we have a simplicial mesh of the continuous object. In this case, we cannot use the dimension constraining method.

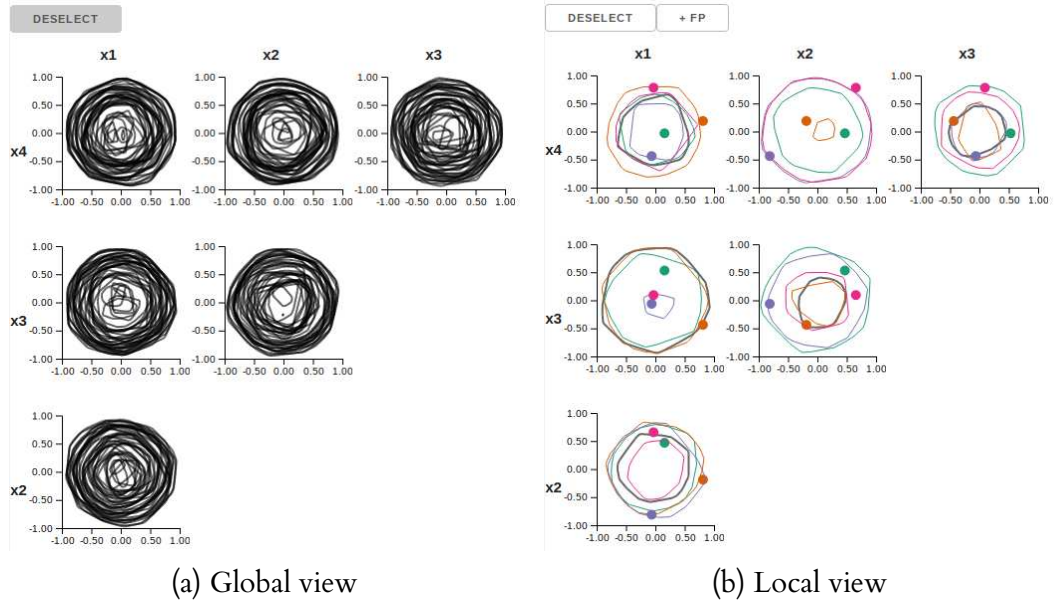


Figure 3.1: The interface for browsing slices created by the Hypersliceplorer algorithm. We show a plot for each pair of dimensions laid out in the same way as HyperSlice [119]. The interface has two modes: global and local view. The global view (a) shows the results of sampling over a number of focus points. The views are linked through highlighting a slice. The local view (b) shows a single selected slice and then the user can add additional slices by clicking the “+ fp” button.

In this chapter, I develop an algorithm for generating two-dimensional slices of these multi-dimensional simplicial objects. This chapter is based on joint work with Torsten Möller, Michael Sedlmair, and R. Mike Kirby currently under submission to the 20th EG/VGTC Conference on Visualization (EuroVis 2018).

3.1 Motivation

The visual analysis of multiple dimensions is one of the central themes of visualization research. In principle there are two conceptual types of problems that amount to two different mental models. (1) Often, the data set is considered to be truly discrete and projection methods, such as scatterplots and dimensionality reduction techniques, are used for its analysis. Typical examples include business applications, in which one is analyzing customer data. The focus of my work is different in that (2) I am focusing on continuous multi-dimensional data spaces. For computational purposes, the data set is then merely a set of points sampled from a continuous phenomenon of study. This is rather common in simulation and engineering applications or for the study of continuous algorithmic parameters in modeling environments, including machine learning applications [98]. Of course, for such scenarios, projection based visualization might be of help as well. However, they do not respect the mental model of the object of study [115].

To comprehend these continuous data spaces, I extend the HyperSlice [119] method, which presents a number of slices through data space, all connected through one point in data space (called the focus point). Slicing has a number of advantages including undistorted views of the space and the preservation of distances. The disadvantage is that only one focus point can be shown at a time. Vastly different views of the object may be seen depending on the location of the focus point. Navigating multiple dimensions, the user may also lose their place during interaction. I create a two-dimensional slice by constraining all but two parameters to the focus point value.

In Sliceplorer [113], I suggest to present 1D slices instead of 2D slices in such cases. While a 1D slice carries less information than a 2D slice, they could now present a

global view of the multidimensional object by over-plotting many 1D slices. This advantage was worth the loss of 2D information. Here, I take the idea of more global overviews and revisit 2D slices for closed multi-dimensional objects, whose overall multi-dimensional shape is of great importance. This has been motivated by a number of real-world application scenarios, from comprehension of multi-dimensional polytopes by geometers, to applications in computational science, to multi-dimensional Pareto-front analysis. Polytopes are the generalization of polygons and polyhedra to multiple dimensions. By showing only slices of the outlines of these polytopes we can again create global views of these data sets through over-plotting of many focus points.

I address the issue of selecting a focus point by sampling a number of focus points and producing projections of 2D slices (Figure 3.1). This *global view* gives the user an overview of the behavior of the object without having to navigate manually. Since we are viewing just the outer hull of the object, I draw these as a projection of a set of 2D slices. I use linked highlighting to show all slices for a particular focus point. In addition, the user can click on a particular slice and switch to the *local view*. The local view begins by showing the particular slice the user clicked on. The user can add additional focus points and select particular slices for comparison. This interaction models Shneiderman’s mantra: “overview first, filter and zoom, details on demand” [103].

My major contribution is an algorithm called Hypersliceplorer for computing the intersection of a 2D slice with a simplicial mesh in any dimension (see section 3.3 and Figure 3.2). The issue is that a 2D plane does not have a well-defined normal in spaces higher than three. Therefore, one cannot use the typical point-normal form of a plane in order to compute the intersection of the plane with the simplex. Instead, I treat the plane as a point with two free parameters representing the plane. Then, I show how this representation allows us to compute how a multi-dimensional simplex intersects a 2D plane. This approach lets one compute slices of a multi-dimensional object without a parametric form of the surface. I also demonstrate the results of this algorithm with an interactive interface I developed.

I evaluate our algorithm and interface in two ways representing their recom-

mended evaluation methods according to the nested model [83]. For the overall technique and interactive viewer, I demonstrate the effectiveness of Hypersliceplorer by presenting three case studies. For the underlying algorithm I present an analysis of the running time.

In summary, my contributions are:

- an algorithm for computing 2D slices of multi-dimensional polytopes defined by a simplicial mesh,
- an interactive viewer combining both global and local views of slices,
- three case studies demonstrating the technique,
- and an analysis of the running time of the slicing algorithm.

3.2 Related work

Multi-objective optimization and multi-dimensional objects are two areas where it is important to study shapes in over three dimensions. I discuss these areas below. Topological techniques are based on viewing critical points of manifolds [24, 36] or how contours merge and split [21]. I do not discuss them further here. Manifold analysis is very different than visualizing shapes and this is discussed chapter 2.

The need to understand multi-dimensional polytopes is apparent to geometers [126]. However, there are a number of cases in computational science where the understanding of the size and the shape of a sub-section of the parameter space is of importance [11, 98]. One of these cases is highlighted in subsection 3.5.2. Another use case is the study of multi-dimensional Pareto fronts (subsection 3.5.3).

3.2.1 Multi-objective optimization

In multi-objective optimization there are several scalar values that we wish to optimize. The set of optimal points is known as the Pareto front. If each objective measure is continuous then we have a continuous hull in one orthant. We want to use this hull

to analyze the trade-offs between objective measures. Interactive decision maps [75] show a 3D Pareto front as a series of 2D slices. Any objectives past three must be constrained to a value however. Objective functions are difficult to sample since we often do not have control over the sampling of the range of a function. To generate this hull one often samples the objective functions and then computes the Pareto points using an algorithm such as NSGA-II [26] or the skyline algorithm [14]. We can then generate the hull using multi-dimensional marching cubes [13], the quickhull algorithm [8], or alpha shapes [31]. These can then be viewed in Hypersliceplorer as I do in subsection 3.5.3.

An alternative is to treat the samples as a fixed set and then visualize the relationship between possible combinations of objectives. Typically this is done by examining the weight space through interaction. LineUp [40] uses a ranked list approach and shows the user how rankings will change as the user changes the relative weighting for each objective. WeightLifter [87] extends this by also showing the stability of rankings. The user can understand how much a particular objective is affected by its weighting. This can help speed interactive exploration. Finally, the joint contour net [18] can be used to compute how often two objectives hold particular values simultaneously. In my case, the mental model is a continuous one. Thus it makes more sense to show a continuous Pareto front.

3.2.2 Multi-dimensional objects

When speaking of 3D polytopes, their source is usually either from reconstruction of 3D point clouds (see Dey [28]) or from iso-surfacing techniques (see Wenger [122]). There are extensions to iso-surfacing techniques in multiple dimensions [13], but in more than three dimensions we must distort the space somehow to visualize the object.

For the visualization of 4D polytopes, there are a number of techniques for moving from four to three dimensions. The Schlegel diagram [108] is one such method based on projection. We pick a face of the figure, usually the largest, which is a three-dimensional object. Then, all other faces are “packed” inside this face in such a way

that we can show the connections between faces. The Schlegel diagram works well for regular polytopes where we have some previous intuition about the faces. However, for an arbitrary simplicial mesh, any face is a simplex which we need to project into. All Schlegel diagrams of a simplicial mesh look like a simplex with a number of other simplices inside them. It can be difficult to recover what the original object looks like because the cross section is lost. An alternative approach is to treat the fourth dimension as time and then produce an animation of the evolution of the shape in three dimensions. In this case each frame of the animation is a 3D slice of the object. Rather than first projecting from 4D to 3D and then rendering the projection, Hanson and Cross [44] propose a method to first render the object in 4D and then view the three-dimensional projection. This allows them to show unique lighting effects from the 4D surfaces. As with all projection methods, if the user is unaware of the details of the method it can be difficult to build a mental model of the shape under study.

Hasse diagrams [9] are based on showing the connectivity between vertices of an object. These can be seen as network diagrams where the vertices of the figure are the nodes in the graph and the edges of the graph are the edges in the figure. These have a number of layout issues. For visual understanding, humans prefer a 2D planar graph [65]. Good layouts of the Hasse diagram must balance human aesthetic needs like few edge crossings with the geometric interpretation. There are automatic layout algorithms, such as the one by Battista et al. [9], but these do not work in all cases.

For more than four dimensions, projection methods no longer work as well. Techniques based on slicing the space can be extended to any number of dimensions. The techniques to perform this so far, such as HyperSlice [119], HyperMoVal [90], and Sliceplorer [113], are designed to show slices of multi-dimensional manifolds. They produce slices by constraining all but two (for 2D slices) or one (for 1D slices) of the dimensions to the focus point value and then producing a heatmap, contour plot, or function plot. Sliceplorer addressed the focus point issue by sampling over a number of focus points and projecting them down. Exploded view diagrams [63] offer a hybrid method between a 3D volume visualization and slicing. However, they are limited to 3D objects. The global view of Hypersliceplorer is inspired by the idea of examining

cross sections. Hypersliceplorer also has a local view which permits the user to look at a small number of self-selected slices. I developed a method to produce slices based on a simplicial mesh which is very useful given a discretized surface (see Figure 3.2).

3.3 Algorithm

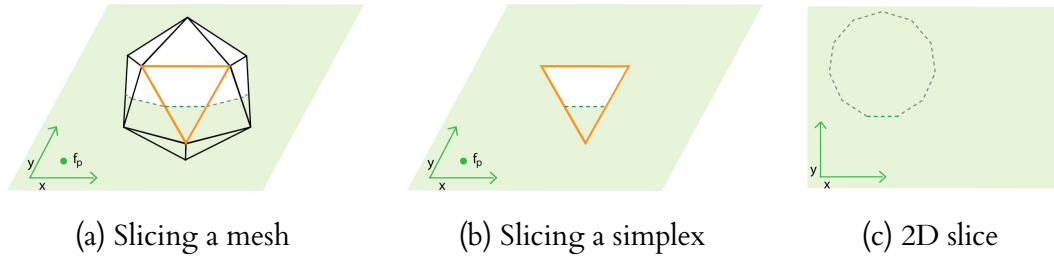


Figure 3.2: An overview of how the algorithm functions. The goal is to compute the intersection of a slice with a polytope defined as a simplicial mesh (a). The slice is defined by selecting a focus point and then extending it in two directions. Each simplex in the mesh (b) is treated independently in order to compute the intersection of the simplex with the slice (see Algorithm 1). The collection of all intersections for a particular plane is shown as a line plot (c). This process is repeated over a number of randomly sampled focus points.

There are a several ready-made solutions to creating a number of uniformly distributed multi-dimensional samples (e.g. Sobol sequence [107] or Latin hypercube [77]). These methods are based on ensuring that the distance between sample points is as even as possible. These will be our focus points. Based on this, the main contribution of Hypersliceplorer is an algorithm that computes slices of a multi-dimensional polytope. The algorithm will produce a single, two-dimensional, axis-aligned slice for a single focus point. To produce the full multi-dimensional view this process is repeated for each focus point and dimension pair.

3.3.1 Conceptual overview

Other slicing techniques, like Sliceplorer [113] or HyperSlice [119], show slices of multi-dimensional manifolds. In this case we can fix all but one or two of the parameters, the parameters representing the slice, to a fixed value based on the focus

point. This gives us a two-dimensional function which we can draw as a one- or two-dimensional function plot.

When we have a simplicial hull instead of a functional description then we cannot use the parameter fixing method. This hull can be computed as a convex hull of a point cloud using an algorithm such as quickhull [8] or they can be pre-defined. In either case, a simplicial hull is a set of $d - 1$ -dimensional simplices for a d -dimensional space. A two-dimensional slice is equivalent to a two-dimensional plane. So, in order to compute the two-dimensional slice, we need to compute the intersection of a two-dimensional plane with a set of $d - 1$ -dimensional simplices (see Figure 3.2a).

The method often used in graphics for computing a plane/simplex intersection is to represent the plane as a point and normal vector. Then, we find the intersection by checking, for each pair of points, if they are on opposite sides of the plane. This works for 2D planes (slices) in 3D space since the normal to the plane is unique. However, in more than three dimensions we cannot use this method to compute the intersection of a 2D planar object. This planar object does not have a well-defined concept of a “side” in a multi-dimensional space. The analogy to this is a line in three-dimensions.

The solution to this problem relies on two key observations. The plane can be represented as a point with two free parameters and then see if this point lies on the boundary of the simplex using barycentric coordinates. A barycentric coordinate defines the location of a point on the face of a simplex. For a single focus point, choose a point somewhere in the domain. Then, select two axis-aligned directions and create a plane by setting the values of that point to free parameters, denoted x and y . These free parameters create a 2D plane. The next step is to see where this plane intersects the simplex (Figure 3.2b). A point is located on the boundary of a simplex whenever at least one barycentric coordinate is zero and the rest are between zero and one. If there is a solution then there will be a formula for the line segment through the simplex (Algorithm 1). These line segments are computed for each plane/simplex intersection. The collection of intersecting lines forms the image of the slice on the plane (see Figure 3.2c).

The simplicial hull of a shape consists of $d-1$ -dimensional simplices. However, in

order to convert to barycentric coordinates without using a pseudoinverse we need a d -dimensional simplex where every point has a unique representation. Using the focus point as an additional vertex will yield a d -dimensional simplex. This will lead us to a square matrix, which is easy to invert. Any intersections with the extra point are removed at the end of the algorithm.

3.3.2 Algorithm details

The algorithm requires that the shape to be sliced is specified as a set of simplices. Each combination of simplex, pair of slicing dimensions, and focus point is handled independently. These can be looped through, as in the pseudocode (see Algorithm 2), or in parallel, as in the actual implementation.

To illustrate how the slicing algorithm works I first introduce some notation. We begin with a d -dimensional focus point f_p and a simplex s consisting of $d + 1$ d -dimensional points, x_1, \dots, x_{d+1} . The slice is denoted as f'_p in the formulas below. Without loss of generality, I will assume the slicing dimensions to be $(d_1, d_2) = (1, 2)$. Then, the two free variables for the specification of the slice, x and y , will replace the first two components of the focus point (Equation 3.2). Let T be the matrix to convert a point from barycentric coordinates to Cartesian coordinates (including a homogeneous component of 1). The columns of T are the $d + 1$ points defining the simplex. The final row of ones ensure that the barycentric coordinates sum to one. The inverse of T will convert a point from Cartesian coordinates (including a homogeneous component of 1) back to barycentric coordinates.

$$f_p = [p_1, p_2, \dots, p_d, 1] \quad (3.1)$$

$$f'_p = [x, y, p_3, \dots, p_d, 1] \quad (3.2)$$

$$T = \begin{bmatrix} x_{1,1} & x_{2,1} & \cdots & x_{d+1,1} \\ x_{1,2} & x_{2,2} & \cdots & x_{d+1,2} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1,d} & x_{2,d} & \cdots & x_{d+1,d} \\ 1 & 1 & \cdots & 1 \end{bmatrix} \quad (3.3)$$

$$(3.4)$$

The next step is to convert the slice, f'_p , to barycentric coordinates, λ .

$$T^{-1} = \begin{bmatrix} \alpha_{1,1} & \alpha_{2,1} & \cdots & \alpha_{d+1,1} \\ \alpha_{1,2} & \alpha_{2,2} & \cdots & \alpha_{d+1,2} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{1,d+1} & \alpha_{2,d+1} & \cdots & \alpha_{d+1,d+1} \end{bmatrix} \quad (3.5)$$

$$\lambda = T^{-1} f'_p \quad (3.6)$$

$$= \begin{bmatrix} \alpha_{1,1}x & +\alpha_{2,1}y & +\alpha_{3,1}p_3 & +\cdots & +\alpha_{d+1,1} \\ \alpha_{1,2}x & +\alpha_{2,2}y & +\alpha_{3,2}p_3 & +\cdots & +\alpha_{d+1,2} \\ \vdots & & & & \\ \alpha_{1,d+1}x & +\alpha_{2,d+1}y & +\alpha_{3,d+1}p_3 & +\cdots & +\alpha_{d+1,d+1} \end{bmatrix} \quad (3.7)$$

This equation is essentially a linear equation in x and y and we denote its coefficients by λ_x , λ_y , and λ_c respectively. Thus, each component is an equation of a line (t is denoting the transpose).

$$\lambda_x = [\alpha_{1,1}, \alpha_{1,2}, \dots, \alpha_{1,d+1}]^t \quad (3.8)$$

$$\lambda_y = [\alpha_{2,1}, \alpha_{2,2}, \dots, \alpha_{2,d+1}]^t \quad (3.9)$$

$$\lambda_c = \left[\sum_{i=3}^{d+1} \alpha_{i,1} p'_i, \dots, \sum_{i=3}^{d+1} \alpha_{i,d+1} p'_{d+1} \right]^t \quad (3.10)$$

$$\lambda = \lambda_x x + \lambda_y y + \lambda_c \quad (3.11)$$

Here, x and y correspond to the horizontal and vertical coordinates of the intersection of the plane with the simplex. Each component of λ reflects the influence of one of the $(d + 1)$ points of the simplex. If the influence is zero (i.e. for the i -th point we have $\lambda_i = 0$) then we are on the boundary of the simplex. An intersection of a plane with a simplex must cross the boundaries (Figure 3.2b) so it suffices to turn each component of λ , λ_i , to zero in turn. It is possible that the plane will intersect multiple faces. Each component of λ is set to 0 in turn the range of x and y in the remaining components are checked to see if they are valid barycentric coordinates (i.e. are between zero and one). If this is the case, then the plane intersects the simplex. Otherwise, there is no intersection. In other words, if we are trying to find the intersection for face i , we need to solve $\lambda_i = 0$ such that $\forall j \neq i, 0 \leq \lambda_j \leq 1$. This can be solved either with a linear constraint solver or directly by solving for y setting $\lambda_i = 0$ and substituting it into each row j of Equation 3.7 and then finding an interval for x and y that allows each λ_j to be between 0 and 1. This can be done by individually finding the valid x, y interval for each j and then taking the intersection of all the individual intervals. If the intersection is non-empty then this is the interval of values for x in the intersection. The x values can be substituted into the formula to find the interval of values for y . These intervals are drawn as line segments on the screen (Figure 3.2c).

Algorithm 1 Slicing a single simplex

```

function SLICE( $p, s, d_1, d_2$ )
     $T \leftarrow [s \ 1]^t$  ▷ from barycentric to Cartesian coordinates
     $r \leftarrow p$ 
     $r[d_1, d_2] \leftarrow [x, y]$ 
     $\lambda_x x + \lambda_y y + \lambda_c \leftarrow T^{-1}r$  ▷ convert to barycentric coordinates
     $\text{rng}_x \leftarrow [-\infty, \infty]$ 
     $\text{rng}_y \leftarrow [-\infty, \infty]$ 
    for  $i \leftarrow 1$  to  $d + 1$  do ▷ each face of the simplex
         $(\text{rng}'_x, \text{rng}'_y) \leftarrow \text{SOLVE}(\lambda_{x,i}x + \lambda_{y,i}y + \lambda_{c,i} = 0, \text{s.t.} \forall j \neq i, 0 \leq \lambda_{x,j}x + \lambda_{y,j}y + \lambda_{c,j} \leq 1)$ 
         $\text{rng}_x \leftarrow \text{rng}_x \cap \text{rng}'_x$ 
         $\text{rng}_y \leftarrow \text{rng}_y \cap \text{rng}'_y$ 
    end for
    return  $(\text{rng}_x, \text{rng}_y)$ 
end function

```

Algorithm 2 Finding slices for all simplices

```

for  $d_1 = 1$  to  $d - 1$  do
  for  $d_2 = d_1$  to  $d$  do                                ▷ all pairs of dimensions
    slices  $\leftarrow [\emptyset, \emptyset, \emptyset, \emptyset]$       ▷ 4 column matrix for min/max  $x$  and  $y$ 
    for  $p \in FP$  do                                       ▷ all focus points
      for  $s \in S$  do                                       ▷ all simplices
        ranges  $\leftarrow \text{SLICE}(p, s, d_1, d_2)$ 
        if ranges  $\neq \emptyset$  then                        ▷ add new row if we found an intersection
          slices  $\leftarrow \begin{bmatrix} \text{slices} \\ \text{ranges} \end{bmatrix}$ 
        end if
      end for
    end for
  end for
  PLOT(slices,  $d_1, d_2$ )                                ▷ plot slices to proper subplot
end for

```

3.4 Interface

I developed an interactive viewer to browse and select slices of interest in order to build up an understanding of the object we are viewing. Slicing is an inherently interactive operation. Depending on what focus point we will see different aspects of the data. In a multi-dimensional space, it is easy to get lost navigating freely without guidance. However, if we show all slices at once the user cannot closely examine one particular aspect of the data. Thus, the interactive interface, shown in Figure 3.1, has two modes: a global view and a local view. The global view is designed to give an overview of the general shape. By selecting a slice and corresponding focus point of interest, the user can then switch to the local view and gradually add additional slices at new focus points.

3.4.1 Global view

The global view (Figure 3.1a) gives an overview of the possible cross sections of the object. By default we show slices for the first 50 focus points sampled using a Sobol sequence [107]. This has the advantage of being both space-filling and easy to add additional sample points if required. Since we are slicing hulls of simplicial meshes,

each slice is a contiguous line plot in the view. We use alpha blending in order to show the distribution of hull shapes in each pair of dimensions. From this the user can get insight into whether or not a shape has a regular structure.

With more than one slice one cannot easily tell how the slices correspond between panels in the layout. I address this by using linked highlighting between the plots. If the user mouses over a slice in one plot the slices corresponding to that focus point are highlighted in the other plots. In addition, the user can click on a particular slice of interest to focus in on that particular slice. This brings the user into the local view mode.

3.4.2 Local view

The local view mode of the interface allows the user to narrow in on a particular focus point and then explore how other slices of the figure relate to that one. The focus point is represented as a dot projected on each sub plot. The user can change the focus point by dragging the focus point dot to a new location. Thus, the user can change one or two focus point values per dragging interaction. The user can also add additional focus points by clicking the “+ fp” button in the upper left of the interface. Each focus point is automatically colored based on a discrete color map from ColorBrewer [45]. The slices themselves and the focus points are linked through a similar color. For example, one mode of exploration this view supports is examining the faces orthogonal to one of the slices. The user can return to the global view by clicking the “deselect” button on the top left of the interface.

3.5 Case studies

Three areas where Hypersliceplorer can be used is in the visual analysis of multi-dimensional polytopes, differences between spaces, and multi-objective optimization. I examine each of these in turn in the sections below.

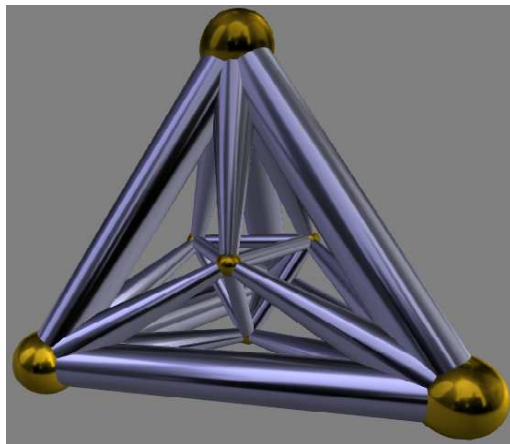
3.5.1 Polytopes

Polytopes are the generalization of polygons and polyhedra to any number of dimensions. Naturally, in more than three dimensions we have no way of viewing these objects directly. Common ways to view them is either through projections, such as the Schlegel diagram [108], or as a graph representation, like the Hasse diagram [9]. Projection methods do not accurately show distances or angles. These must be distorted to show a multi-dimensional object in two or three dimensions. Network diagrams do not necessarily show symmetries or structure unless care is taken during layout.

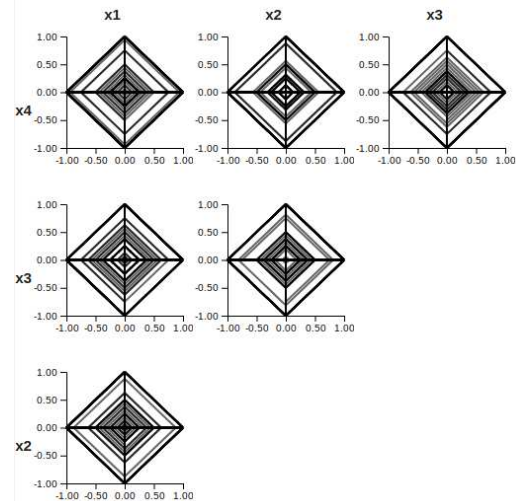
As an alternative, we can slice these polytopes and examine the slices. Regular polytopes have well-studied structure and symmetries so I use these as verification examples. I examine these with Hypersliceplorer and see if the structure matches reality. For example, in Figure 3.3 I show a 16-cell which is the four-dimensional version of an octahedron in both Hypersliceplorer (Figure 3.3b) and as a Schlegel diagram (Figure 3.3a) using the Stella4D software [121]. The Schlegel diagram picks a face of the 16-cell and projects the remaining structure inside it. In this case, each face of the 16-cell is a simplex. From the Schlegel diagram it is difficult to see that the dual of the 16-cell is the hypercube (i.e. each vertex of the hypercube corresponds to a face of the 16-cell and vice versa). However, in Hypersliceplorer this property is clear from looking at the cross sections. We can see that each cross section looks like a rotated cube which comes from the dual property. In addition, we can see the simplicial faces from the horizontal and vertical lines in the view. These result from intersections of the 2D slice with a face of the 16-cell directly.

Further, Hypersliceplorer allows us the visualization 3D or 5D analogs of the 16-cell (the octahedron in 3D, Figure 3.3c, as well as the 5-orthoplex in 5D, Figure 3.3d). The Schlegel diagrams cannot be scaled to higher dimensions.

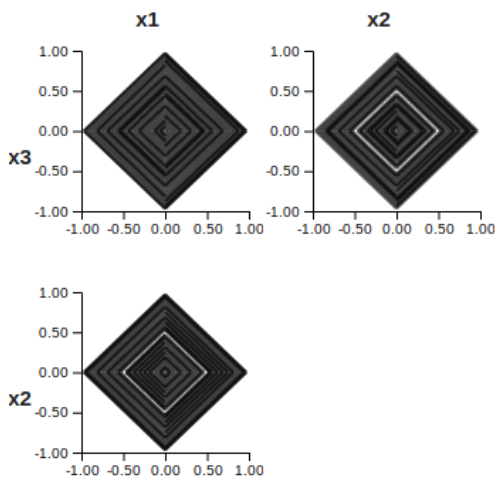
We can also look at other regular polytopes in the same fashion. Figure 3.4 shows a hypercube in 3-, 4-, and 5-dimensions. From these plots we can clearly see the generalization of the square, to the cube, to higher dimensions. One of the advantages



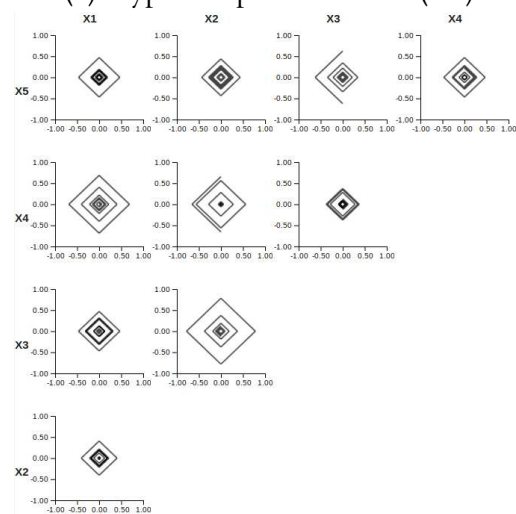
(a) Schlegel diagram: 16-cell (4D) generated using Stella4D [121]



(b) Hypersliceplore: 16-cell (4D)



(c) Hypersliceplore: Octahedron (3D)



(d) Hypersliceplore: 5-orthoplex (5D)

Figure 3.3: This figure shows the 16-cell, the 4-dimensional regular orthoplex (4D version of an octahedron) as (a) a Schlegel diagram and (b) the Hypersliceplore view. The Hypersliceplore view shows the outside shape of the figure and the repeating structure. We can also see the repeating structure in the 3D (c) and 5D (d) views.

of my method is that I do not need to choose a face to project into. For example, with a discretized hypersphere, there are many faces. We can see in Figure 3.5 the regular cross section of a sphere as well.

3.5.2 Positive and Bernstein polynomials

In physics, it is sometimes necessary to fit data with a function that is positive everywhere on its domain. An example of such data is density; it is a positive quantity and any regression on it should be positive. In numerical methods, we often use poly-

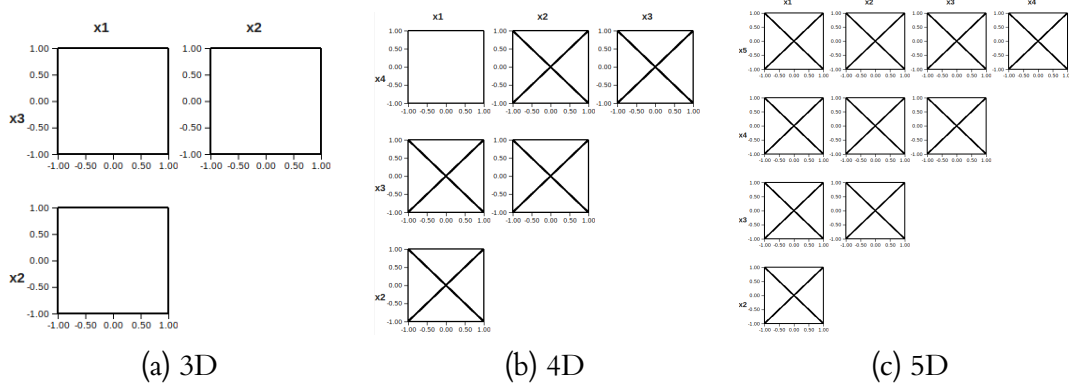


Figure 3.4: 3-, 4-, and 5-dimensional hypercubes. We can see the regular structure in the cubes. The cross sections are all the same size since the cube is oriented to the axes. The cross lines in the plots are due to the simplicial mesh.

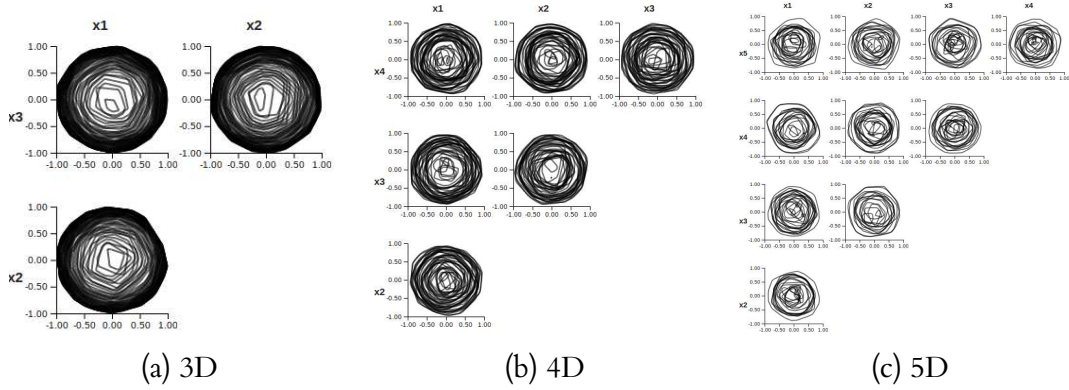


Figure 3.5: 3-, 4-, and 5-dimensional hyperspheres. We can see the concentric rings from slicing the sphere at different points. The irregularity of the slices is due to sampling.

nomials as a means of representation, and hence we want to find polynomials that are positive on some compact domain, without loss of generality say, $[0, 1]$. It is very difficult to control a polynomial such that it is positive during the fitting process using a linear constraint solver. One method used by physicists is to restrict the fitting process to Bernstein polynomials [89]. By only using positive Bernstein coefficients, Bernstein polynomials are restricted to be strictly positive. However, physicists do not know how “representative” are the positive Bernstein expansions of the space of positive polynomials. In other words, can every positive polynomial be represented by a corresponding Bernstein polynomial? To show these differences, I select a large number of polynomials and visually compare the spaces in order to understand the differences.

A Bernstein polynomial of degree n is a linear combination of $n + 1$ basis poly-

mials.

$$B_n(x) = \sum_{i=0}^n \beta_i b_{i,n}(x)$$

where β_i is a scalar factor and $b_{i,n}$ is a Bernstein basis polynomial. These are defined as,

$$b_{i,n}(x) = \binom{n}{i} x^i (1-x)^{n-i}.$$

Each of the Bernstein basis polynomials is positive in the range $[0, 1]$, therefore if we constrain all $\beta_i \geq 0$ then the resulting polynomial will also be positive.

Sampling method

The space of a polynomial of degree n is the range of each of its coefficients. For example, a 2nd degree polynomial, $a_0 + a_1x + a_2x^2$, has three coefficients: a_0 , a_1 , and a_2 . Since we are only concerned with positivity, any polynomials that differ from each other by a positive factor are equivalent. Thus, the polynomials $0.2x^2 + 0.1x + 2$ and $0.1x^2 + 0.05x + 1$ will be positive in the same range. Therefore, I constrain the sampling by setting one of the coefficients to 1 or -1 and then sampling the rest between -1 and 1. I used 10,000 sample points to get a representative sample. Since one coefficient is constrained in turn to either -1 and 1 there are $2n$ possible polynomials where the remaining n coefficients range between $[-1, 1]$. I then test to see if each polynomial is positive in the domain $[0, 1]$. I also determine if the equivalent Bernstein polynomial has all $\beta_i \geq 0$. We can compute the β_i factors because each coefficient a_j is a linear combination of Bernstein coefficients. For our two-dimensional example: $a_0 = \beta_0$, $a_1 = 2(\beta_0 - \beta_1)$, and $a_2 = \beta_0 - 2\beta_1 + \beta_2$. Then, for each polynomial, it is either non-positive, positive without positive Bernstein coefficients, or positive with positive Bernstein coefficients. I perform this process to examine polynomials of degree 3, 4, and 5. Constraining a coefficient to ± 1 produces a face of the space of polynomials. These faces are convex so I can generate a convex hull of these points and examine them using Hypersliceplorer. Since we are only interested in how these spaces differ, we examine the difference views.

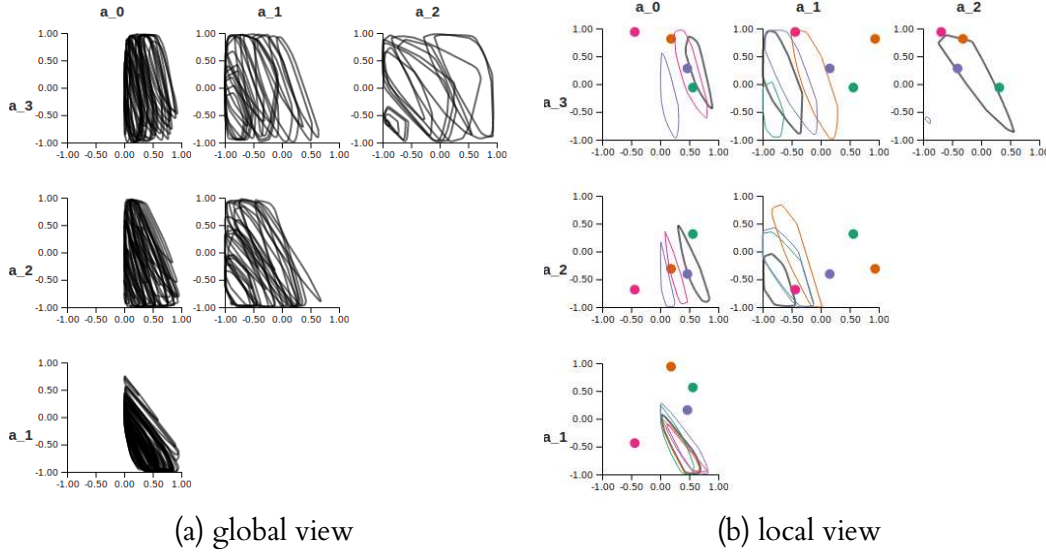


Figure 3.6: The difference between possible coefficient values for general positive polynomials, $a_0 + a_1x + a_2x^2 + a_3x^3 + x^4$, and polynomials that can be represented with positive Bernstein coefficients. From the global view (a) we can see that the area of the slices is quite large. This means that the difference between spaces is quite large, especially with respect to the higher-order coefficients. We can narrow into a particular slice in the local view (b) we can see the orthogonal faces to the slice in the $a_2 \times a_3$ plot.

The current assumption is that, while there is a difference between the set of all positive polynomials and the set of all Bernstein polynomials with positive coefficients, that difference is small. However, the Hypersliceplorer visualization shows that this is not the case. Figure 3.6a shows all 4th degree polynomial with the x^4 term fixed to 1. We can see that for almost an entire range of the x^2 and x^3 coefficients a_1, a_2 there are positive polynomials for which one cannot find a Bernstein polynomial with positive Bernstein coefficients. Using the local view (Figure 3.6b), we can see that this difference is also large in the other dimensions. Other patterns become apparent, such that $a_0 > 0$ which could lead to novel hypothesis that can be tested.

The global overview also lets us compare across degrees of polynomials. In Figure 3.7 I show a 3rd, 4th, and 5th degree polynomial with the coefficient of the x^2 term set to 1. Here we can see that the $a_0 \times a_1$ plots all look the same. In fact, the width across all the panels including a_0 are the same. In this case this means that for these ranges of a_0 (the constant term) we will not be able to find a Bernstein polynomial with positive Bernstein coefficients no matter the degree of polynomial.

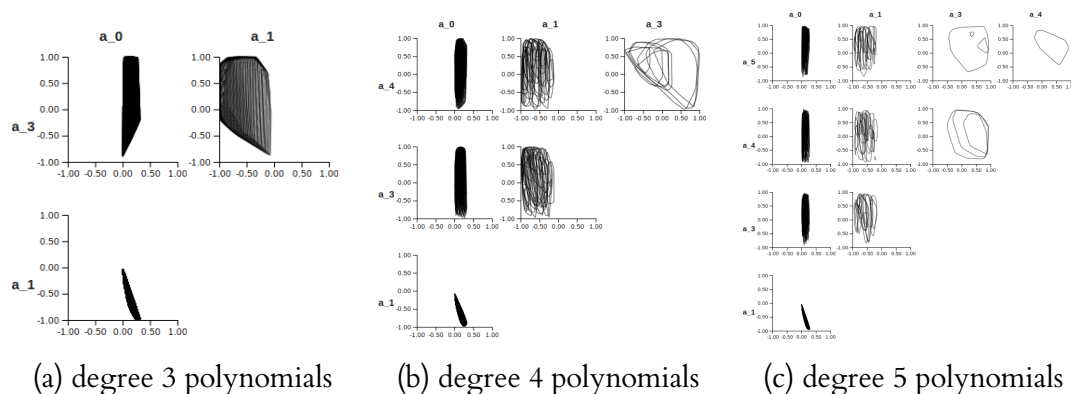


Figure 3.7: Examining differences in the space of general positive polynomials and Bernstein polynomials with positive Bernstein coefficients. In this example the x^2 term is set to 1. We can see that across degrees of polynomials, the space differences in the a_0 and a_1 coefficients is relatively consistent. The empty plot in c for the a_3, a_4 plot is because the focus point sampling did not hit a particular slice. The solution is to add additional focus point samples.

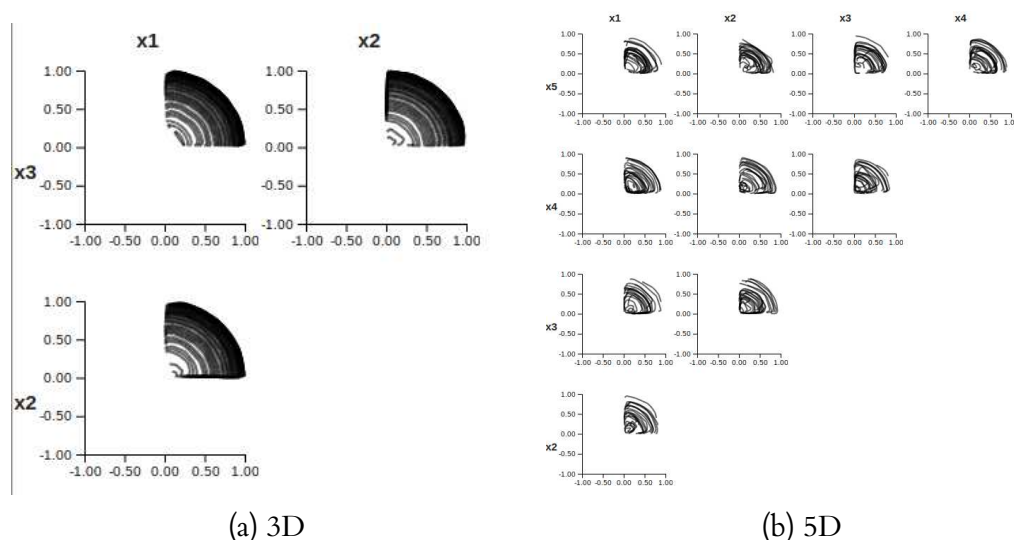


Figure 3.8: Hypersliceplorer views of spherical Pareto fronts in 3D (a) and 5D (b). The smoothest possible Pareto front is the positive orthant of a hypersphere. From the Hypersliceplorer view we can clearly see the concentric arcs. Each arc allows the user to compare the trade-offs between two objectives given that all other objectives are fixed. Changing from one arc to another means changing other objective settings.

3.5.3 Pareto fronts

A Pareto front (also known as the efficient frontier) is the set of all points that are optimal with respect to some trade-off between objectives. Algorithms such as the skyline algorithm [14] or NSGA-II [26] can extract these points automatically. The issue with multi-objective optimization is this trade-off is not always known. Thus visual analysis of the trade-offs is necessary. With two objectives, this can be visualized

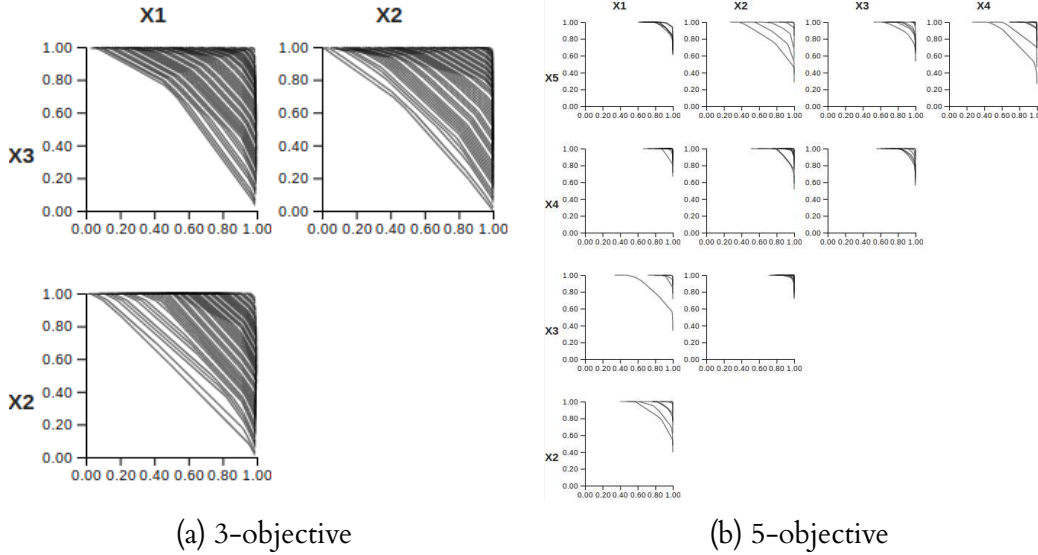


Figure 3.9: Visualization of the 3-objective (a) and 5-objective (b) DLTZ1 problems [27] in Hypersliceplorer. The DLTZ1 Pareto front is a hyperplane that cuts diagonally across the objective dimensions. In the 3-objective case we can see the slices of the hyperplane. The NSGA-II [26] algorithm tends to push points towards one objective. This becomes much more pronounced in higher dimensions (b) where the Pareto front is squared off.

using a scatterplot or line plot of the two objectives against each other.

In more than two dimensions this is no longer a curve, it is a hull. The common technique is to use a scatterplot matrix with discretely sampled points. This, however, hides the trade-offs between points in the other dimensions. Instead, we can examine the hull of the Pareto front by slicing using Hypersliceplorer.

The smoothest possible Pareto front is a sphere in the positive orthant of the objective space. In order to illustrate our technique we show a 3D and 5D positive orthant section of a sphere in Figure 3.8. Each arc is the trade-off holding all other parameters fixed. In the multi-dimensional view, setting the focus point is analogous to fixing all but two parameters. Thus, in one plot one can directly see the trade-off between two of the objective measures given that all other objectives are held in place. However, the user can also see at a glance what are the *possible* trade-off curves for a pair of objectives. This helps the user to understand what are the costs and benefits of changing one of the remaining objectives.

I also show a popular multi-objective test problem, DLTZ1 [27] with 5 objectives. I find the Pareto points using the NSGA-II [26] algorithm. I used the same settings

for the algorithm as in Deb et al. [27]. In real-world situations, the Pareto front is not convex. One can use, for example, alpha shapes [31] to generate a non-convex hull of a set of points. For this example, we use the convex hull of the points generated using the quickhull [8] algorithm since the Pareto front of DLTZ1 is convex. The Pareto front is a hyperplane that cuts diagonally across the objective dimensions. In the 3-objective case (Figure 3.9a) we can see how the NSGA-II algorithm is getting close to fitting the points to the hyperplane. It does not find it exactly though which is why the lines appear bent in the figure. In addition, NSGA-II pushes some points out to the maximum value for each objective. This creates the horizontal and vertical lines at the edges of the plots. This becomes much more pronounced in higher dimensions (Figure 3.9b) where the Pareto front appears as a corner shape.

3.6 Algorithm performance

Table 3.1: Results of timing the Hypersliceplorer algorithm on a number of different datasets. I ran the algorithm setting the number of slices to 1,000. I record the number of simplices created by the quickhull algorithm [8], the total time for all slices, and the time per simplex. The time per simplex is the total time divided by the number of plots (i.e. dimension pairs, the number of simplices, and the number of focus points). The time per simplex is roughly constant.

Dataset	Dims	Simplices	Total time (sec)	Time/simplex (ms)
Cube	3	12	48	1.345
Octahedron	3	8	38	1.624
Sphere	3	596	1,243	0.696
Tesseract	4	58	289	0.833
16-cell	4	16	108	1.135
3-sphere	4	2,567	14,283	0.927
5d-cube	5	316	2,378	0.753
5d-ortho	5	32	258	0.808
4-sphere	5	12,886	130,453	1.012
Klein bottle	4	36,258	129,158	0.594

In order to test the running time of the slicing algorithm I ran a number of experiments to understand the timing. I tested regular polytopes in 3, 4, and 5 dimensions

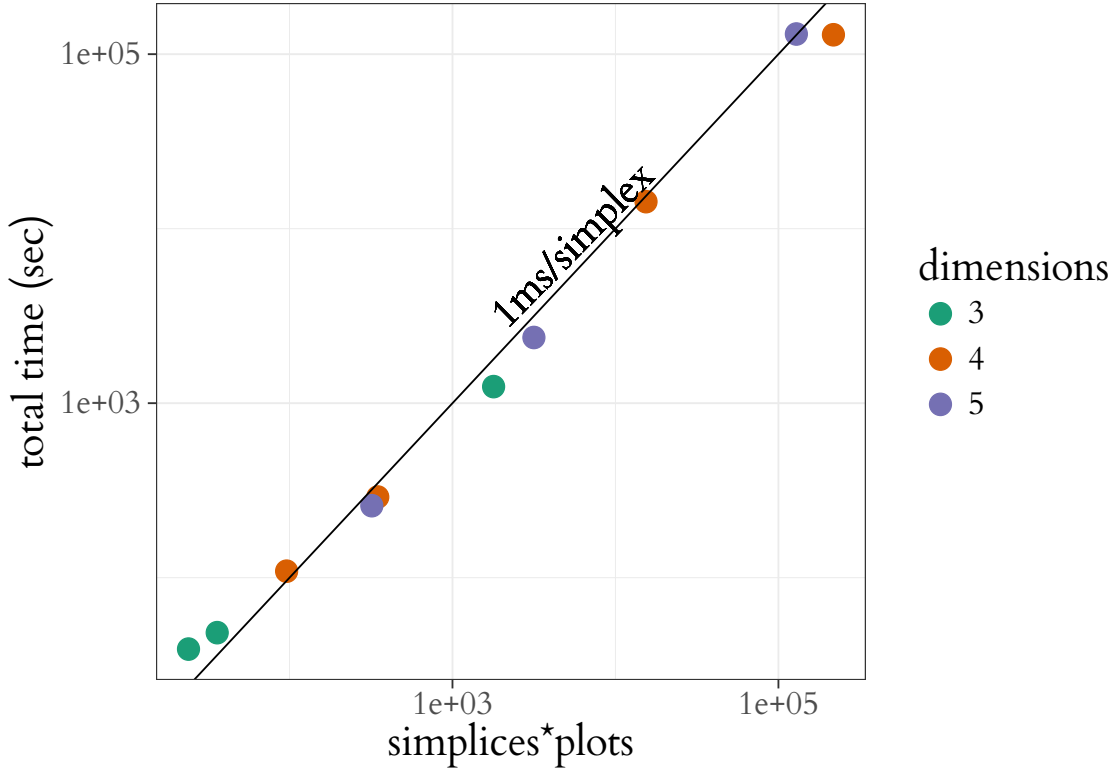


Figure 3.10: Chart showing the number of slicing operations (simplices \times number of plots) versus timing results from running our algorithm on a number of different datasets. The axes are on a log-log scale. The points are all clustered around the “1 ms/simplex” line showing that the running time is roughly one millisecond per slicing operation.

as well as hyperspheres. I also tested the four-dimensional version of the Klein bottle because it has a large number of simplices in its mesh. For each test, I ran the slicing algorithm for all pairs of dimensions and for 1,000 focus points. I recorded the total wall clock time as well as the number of simplices given by the quickhull algorithm. The testing machine has an 8-core 3.2GHz Intel i7-6900K with 64GB of RAM.

The results are shown in Table 3.1. The total number of slicing checks (Algorithm 1) is the number of simplices, times the number of pairs of dimensions ($\binom{d}{2}$), times the total number of focus points (1,000). I divide the total time by this number to show the time per simplex.

As we can see, the times are roughly constant between the number of dimensions and simplices (see Figure 3.10). The reason the Klein bottle is faster is because many slices do not hit any simplices and the algorithm will exit early once this is detected. Right now this algorithm is not optimized. For example, it would be greatly beneficial

to pre-compute a spatial data structure so that only slices that are likely to intersect simplices are evaluated. Currently, the algorithm must check every simplex against every focus point for every pair of dimensions. This is a lot of extra work for figures such as the Klein bottle with many simplices.

3.7 Conclusion and future work

In this chapter I have presented Hypersliceplorer, an algorithm to compute 2D slices of multi-dimensional shapes defined as a simplicial mesh. I also discussed an interactive interface we developed to view the slices. I evaluated our method in two ways: through three target use case scenarios and with measuring the running time.

In the future I will improve the speed of our algorithm by integrating a spatial data structure such as a k-d tree or bounding box method. The current algorithm must find an intersection with every simplex in the figure even if no intersection is possible. The data structure will help to avoid these extra checks. This should improve the speed of the algorithm by avoiding unnecessary intersection tests.

Currently, I have only tested our method with convex hulls of shapes. I plan to also examine the visualization possibilities with non-convex hulls and with pre-generated hulls. Perhaps this method will lead to new insights in mesh generation algorithms.

I will also work closely with target user groups to customize the interface for their specific goals. For example, geometry users may more integration with the Schlegel diagram while multi-objective optimization users may need better support for local neighborhoods.

4 | Fast slicing

With slicing, the user should be able to interactively set the focus point to any position they desire. If we cannot update the view quickly enough then this will break the cognitive connection between the mental model of the user and what they are viewing on screen [101]. The multi-dimensional structures we are visualizing often have some regular geometric structure like simplices or spheres. In this chapter I examine how we can take advantage of the power of the parallelism of the graphics processing unit on any modern computer to create interactive-time focus point selection of multi-dimensional objects [111]. This chapter is based on work published as Thomas Torsney-Weir, Steven Bergner, Derek Bingham, and Torsten Möller. Predicting the interactive rendering time threshold of Gaussian process models with HyperSlice. *IEEE Transactions on Visualization and Computer Graphics*, 23(2):1111–1123, February 2017.

4.1 Motivation

Many scientific studies investigate the relationship between several explanatory variables (inputs) and one or more system response variables (outputs), thereby leading to multi-dimensional data sets. Such data can arise in exploration of the input-output map for applications ranging from weather, physics and biological processes to image segmentation systems. In these cases, the output is actually a complex object such as a segmented image or 3D+time weather data. A key step towards learning about the mechanisms that are present in a computational model or laws that govern natural phenomena is to study how changes in the input variables affect the output. Visual

inspection of individual outputs is suitable in small multiples, but does not scale well with increasing numbers of parameters, because of the large number of runs that are required to adequately represent model behavior in the region of interest. To more comprehensively compare outputs, automation can be taken a step further, for instance, by processing the outputs with feature extractors or fitness functions that are relevant to the driving questions. An interactive, visual investigation of the resulting feature density distribution or fitness landscape then becomes possible [33, 82, 90], but is subject to some fundamental numerical challenges that are topic of this chapter.

The general approach to study deterministic computer models is known in the statistics community as *the design and analysis of computer experiments* [97]. This method involves reconstructing a continuous functional representation of the relationships among variables of the system from a discrete set of samples and then investigating the input/output relationship of the function. Numerical methods for this purpose include local derivative computation, global sensitivity indices [96], and response surface exploration [15]. However, these derived computations have to be set up carefully to yield meaningful results.

The most well known example of non-interactive visualizations of the relationships is the scatterplot matrix for viewing discrete samples. Another example are continuous plots of “average” behavior over the range of each dimension, as exemplified in Chapman et al. [22]. However, any 2D or 3D view of a multi-dimensional space necessarily requires aggregation of that space. We can only “see” a subsection of the parameter space at one time. Therefore, one must create multiple static views, each looking at the data from a different perspective. A scatterplot matrix, for example, shows a 2D projection of the data for each pair of dimensions.

By allowing for user interaction one is not limited to a predetermined set of views. When the view selection changes then a new view of the data must be built on the fly. However, if the visualization system does not respond quickly to the user’s interaction then the cognitive connection with the visualization is lost [101] along with the advantage of adding interaction in the first place. Arguments about what exact response time makes a visualization *interactive* vary. However, view updates somewhere

between 10fps to 60fps are typically deemed acceptable.

One interactive, multi-dimensional, continuous visualization method is HyperSlice [119], which presents the user with a matrix of 2D slices of a multi-dimensional continuous function around a particular viewpoint in space. HyperSlice allows the user to change the location of the viewpoint around which they are inspecting. Given this method, it would be ideal to know if the number of points or the dimensionality of the dataset will overwhelm the graphics capabilities of the user's machine and slow the frame rate. Hence, we need a way to evaluate a priori what the frame rate will be given some data. The main aim of this chapter is developing a methodology to estimate the rendering time of a multi-dimensional visualization system in the form of a predictive formula. We can even invert this formula so that, given a desired frame rate, we can compute the number of points possible to render in the given time. This inversion can be used, for example, to automatically sub-sample the dataset when the predicted rendering time will be too slow.

In order to be able to *predict* the rendering time we need a function for the average rendering time based on the size of the $N \times d$ multi-dimensional dataset as well as the search distance, r , over all possible view points. The advantage of a predictive formula is that, once fit, one can estimate the rendering time for all unknown values of N , d , and r . In addition, we can use this function to examine the time and accuracy trade-off in terms of point spread versus number of samples.

A proper prediction function should describe the number of pixels that will need to be drawn based on the scene geometry. Adapting this function to each user's hardware platform, requires a universal methodology that can be run on each user's environment to make accurate predictions. Combining this strong theoretical foundation with a fitting step makes our method robust to further developments in GPU technology and algorithm development. One can simply recompute the time it takes the GPU to filter and draw the points without having to worry about hardware-specific optimizations.

The contributions of this chapter are:

- An evaluation of how to render multi-dimensional slices on the GPU and how one can use that to predict the number of pixels drawn on screen.
- A fitting procedure for predicting rendering times on an individual user's hardware.
- An application of the prediction formula where I show an algorithm for subsampling data until I can render interactively. I also show a UI dialog box for selecting the number of samples based on the predicted rendering time.

4.2 Related work

The prediction of rendering times is a staple of the 3D rendering community, (e.g., see early work by Funkhouser and Séquin [34]). However, these are focused on the three-dimensional rendering domain. It has yet to be analyzed in the multi-dimensional domain, which I do here. Another difference with this setting is that I have a known scene geometry that I can take into account. Furthermore, my focus is on the conversion of this high-dimensional data representation to 2D views and not global illumination.

One could perform an iterative search method, for example bisection search, on the number of sample points that one could render in interactive time. However, that would need to be performed for every different combination of dimensionality, number of samples, and search distance (d , N , and r respectively). Furthermore, this bisection search may be prohibitively expensive if we are determining the number of samples of a complex simulation where each sample takes hours or days to compute.

4.2.1 Multi-D visualization

Analyzing multi-dimensional data locally is typically done by constraining each dimension within an interval [102]. Arguably, the most popular method to visually inspect multi-dimensional data is a scatter plot or scatter plot matrix (SPloM). Alternatively, one can use parallel coordinates [58] or some type of radial chart [60].

The Prosecution Matrix [117] allows the user to explore the density of input parameter settings that match certain performance criteria. The user specifies what constitutes suitable output parameters as well as a “tolerance box” which represents the possible range for input settings. The system then shows a number of 2D density plots — one for each pair of parameters — indicating how many of the performance criteria were in compliance. HyperMoVal [90] also relies on the user to define a slab around a particular 2D slice. The sample points within this slab are considered relevant to the view and drawn on screen. The focus of HyperMoVal is visualizing how well a model fits sampled data. Their desire is to show how “close” data points fall to a regression line.

Often, data points represent samples of a continuous function. Hence, it is quite common to reconstruct this function as best as possible from samples and visually inspect reconstructed continuous, multi-dimensional function. In this regard, HyperSlice [119] plots two-dimensional orthogonal slices of a continuous function around a local viewpoint. This allows the user to visually inspect the behavior of the function around this point. One advantage of HyperSlice is that it improves the quantitative means for analysis of our multi-dimensional function at least locally by measuring 2D distances. It is difficult to understand distances in multi-dimensional spaces and 2D has been shown to work better for quantitative understanding than 3D [114].

Tuner [112] uses the HyperSlice method to visualize the effects of each parameter around a particular candidate point. Tuner is focused on finding the optimum parameter settings for a computer simulation subject to a number of criteria. The optimum parameter setting must be “high” in the sense of maximizing the objective function but also “stable” in the sense that changes in the parameter settings will not produce large changes in the objective measure. This local sensitivity analysis is visually supported with a HyperSlice view of the high-dimensional parameter space.

4.2.2 Multi-D interpolation

For estimating a continuous function, a popular technique is kernel regression [104]. In this case, the estimated value at a particular point in the parameter space, x' , is computed by averaging over all sample points weighted by a kernel function. Formally this can be expressed as

$$\hat{f}(x) = \sum_{i=1}^n \varphi(x_i - x') f'(x_i) \quad (4.1)$$

for n sample points, x_i . The function $\varphi(\cdot)$ is an approximating kernel. In this case $f'(x_i)$ is the normalized sample value of the function $f(\cdot)$ at location $x_i \in \mathbb{R}^d$ in parameter space. The normalization factor ensures that we can compute the known values of the sample points. This factor is either automatically computed as in the case of Gaussian process regression or explicitly computed from the local neighborhood.

Often the squared exponential kernel is chosen for $\varphi(\cdot)$. This function has one or more bandwidth parameters which control the amount of smoothing between each sample point. The amount of smoothing also affects the distance at which a data point will have an effect on our regression function. While the bandwidth can be set manually, it can also be set by examining the spatial variation in $f(x)$. The Gaussian process model (GP) [92], uses statistical variation to fit the kernel bandwidth appropriately. In the squared exponential case,

$$\varphi(x_i - x') = \sum_{j=1}^d e^{\theta_j (x_{i,j} - x'_{j})^2}, \quad (4.2)$$

where j denotes the particular value for a certain dimension so $x_{i,j}$ is the value of the j th dimension of sample i . The value x_j is the j th dimension of the prediction point. Therefore, there is a separate parameter, θ_j to fit for each dimension. Another approach, exemplified by Hong et al. [53], is to set the kernel bandwidth to take into account the Voronoi cells around each data point. In either case, I recognize that setting the bandwidth is data-dependent. Therefore, I test rendering performance for a number of different kernel sizes.

4.3 Problem description

One method of studying the input/output relationship of computer simulations is known as the *black-box model*. The black-box in this case refers to the simulation code itself. The simulation code is complex and expensive to run so direct analysis is difficult. Under this analysis method one does not make any assumptions as to the inner workings of the simulation code. Instead, we model the simulator as an unknown continuous function that takes a number of numerical inputs and produces a number of numerical outputs. We know the domain of the inputs. We want to study how varying the inputs affects the output.

While we don't know anything about how the simulation works internally, we can sample it by selecting a particular input setting through the simulation and recording the outputs. We can then use a continuous reconstruction method built from a number of samples in order to estimate the *response surface*. This fitted continuous function is known as an *emulator* in the statistics literature. We can then study the input/output relationship using the emulator instead.

4.3.1 1D analysis example

A common choice in the statistics community for this emulator is known as the Gaussian process model [92]. One advantage of the Gaussian process model is that the form is very well known and easy to analyze. It also allows us to measure the uncertainty of the estimation. In order to illustrate the advantages we will go through a 1D example here using a known function $f(x) = \sin(x) + \cos(2x)$ as a stand-in for some simulation code.

We begin by taking a number of discrete samples of the function. Ideally we take as many as possible but this may be limited in terms of time or budget. Since we do not know anything about the behavior of the function in the region we are sampling, we sample in some uniform random fashion. The function as well as the sample locations are shown in Figure 4.1.

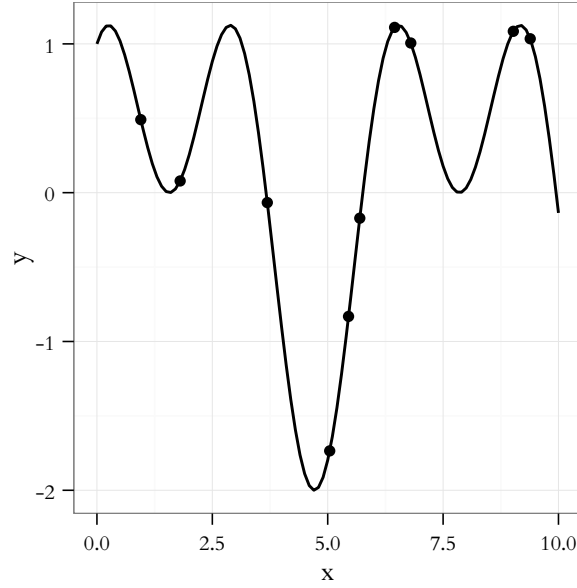


Figure 4.1: An example of taking 10 uniformly distributed samples of the function $f(x) = \sin(x) + \cos(2x)$. The dots show the sampling locations.

We would prefer to take as many samples as possible in order to identify the various peaks and valleys. The interpolation method we choose depends on how we expect the values to change between the sample points. If we expect linear behavior then fitting a piecewise linear function would be ideal. If we expect more complex behavior then we should fit higher-order functions. I show 3 different fitting methods for the function in Figure 4.2: piecewise linear (blue), cubic spline (green), and Gaussian process model (red) along with the true function (black). In this case the cubic spline and Gaussian process model interpolations are very close to the true function, but the true function normally would not be known beforehand.

The basic assumption of the Gaussian process model, however, is that the function behavior between the sample points is random in the sense it could take any path as long as it intersects the points and the correlation function we select gives the general form of the function between the points. The distribution of possible paths follows a multi-variate Gaussian distribution through the selected sample points. The mean of this distribution is the most likely path, which is typically what is visualized. By modeling the behavior this way we also get a model for the uncertainty at any point in the domain. This uncertainty grows in proportion to the distance to the sample

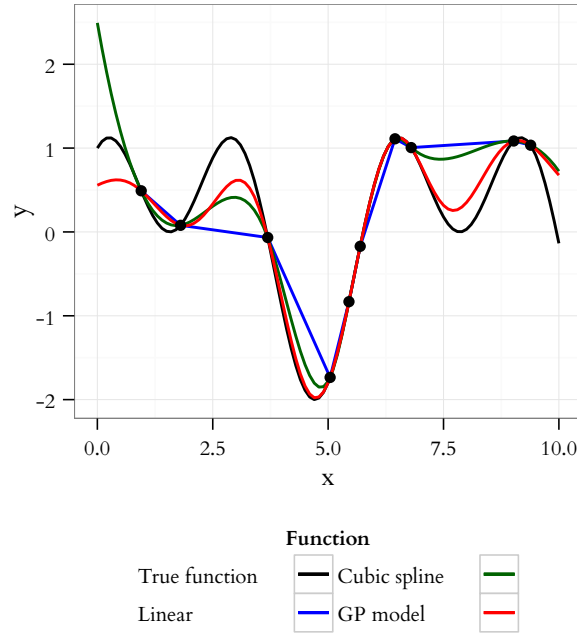


Figure 4.2: Here, I show different interpolation methods of the function $f(x) = \sin(x) + \cos(2x)$ using the same 10 uniformly random distributed sample points. I show the true function as well as piecewise linear, cubic spline, and Gaussian process interpolations.

points. In Figure 4.3 I show the Gaussian process estimation of the above function given the sample points along with the standard error of the estimation. The error grows very quickly when extrapolating, for example when $x < 1$. This is because we are moving away from all sample points. This is why in real applications it is important to sample near the edge of the domain.

Parameterizing a Gaussian process model means correctly parameterizing the correlation functions to the data samples. If the function varies a lot between the sample points then we would expect low correlation between the sample points, while if the function is relatively stable between the sample points then we would expect high correlation between the points. In the spatial sense, this high and low correlation can be seen as the amount of influence the value of a particular sample point has on the value at another location a particular distance away.

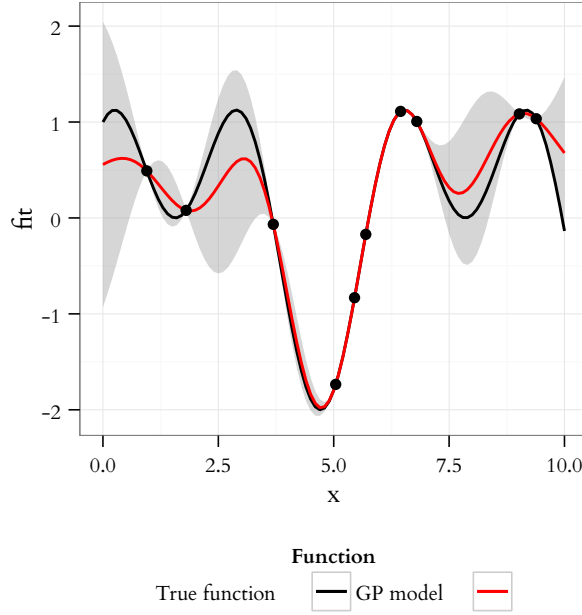


Figure 4.3: The Gaussian process interpolation of the function $f(x) = \sin(x) + \cos(2x)$ from 10 uniformly distributed samples. The standard error of estimation from the model is shown in gray. Note that the standard error increases with the distance from the sample points.

4.3.2 Applications in multi-D

Gaussian process regression is very common in the statistics community to analyze simulations among other types of data. There exist a number of examples where Gaussian process regression along with a uniformly distributed experimental design is used in order to run an analysis. Using uniform sampling with a Gaussian process model has been applied in an optimization scenario, as with Couckuyt et al. [25]. They used a sparse initial sampling and then built a GP model to emulate microwave filter and textile antenna simulations. They then incrementally ran additional samples of the simulation in order to find optimal parameter settings. This process of finding additional sample locations can be quite expensive computationally. Hutter et al. [57] develop a method to find new sample locations when under a time budget. They then applied this method to find optimal parameters for a search algorithm for a propositional satisfiability solver. This method was also used to perform a sensitivity analysis of an arctic sea ice prediction model [22]. They decomposed the Gaussian process

model to find “average” behavior due to each model parameter. Linkletter et al. [71] used Gaussian process models to measure the sensitivity of parameters to a cylinder deformation simulation to reduce the parameter space from 14 input factors to seven. Kaufman et al. [64] use compactly supported correlation functions to build Gaussian process models efficiently on very large data sets. This was applied to cosmological data. Hensman et al. [50] used an approach to train a Gaussian process model on 700,000 data points in an 8-dimensional space to build a model to predict flight delays using a lower rank covariance matrix. Shepherd and Owenius [100] used Gaussian process models as a classification tool in order to classify voxels in dPET images in order to find tumor sites.

Table 4.1: A summary of the literature described in subsection 4.3.2. I show the domain of application of each paper, their analysis goal, as well as the number of samples and number of input parameters (dimensions) of the simulation used to train the GP model.

Reference	Application	Dimensions	Samples	Goal
Couckuyt et al. [25]	Microwave filter	5	51	Optimization
Couckuyt et al. [25]	Textile antenna	2	14	Optimization
Hutter et al. [57]	Propositional logic satisfyability	4	25	Optimization
Chapman et al. [22]	Sea ice prediction	13	157	Sensitivity analysis
Linkletter et al. [71]	Cylinder deformation model	14	118	Sensitivity analysis
Kaufman et al. [64]	Cosmological data	4	20,000	Prediction
Hensman et al. [50]	Flight delays	8	700,000	Prediction
Shepherd and Owenius [100]	dPET data in radiation oncology	4	6 patient images	Classification

As one can see, there are a wide variety of application domains. However, all these analyses share commonalities. I show the summary information of these studies in Table 4.1. The number of inputs is typically on the order of 5–15 inputs. Each of these may correspond to a known factor that can vary in the real world like wind speed or the velocity of a particle, or an unknown fixed-quantity in the real world like Planck’s constant or the gravitational constant. The simulation code typically creates a complex object like a 3D+time model of the world or a segmented image. On these outputs scientists define a number of feature extractors or objective functions which reduce the complex output to a set of numerical attributes [98]. Therefore, for each simulation run we get a vector of scalar input factors and the corresponding vector of scalar outputs. Each scalar output can be considered in a separate analysis so in this

chapter we assume that there is only one scalar output for each input configuration.

The practical number of simulation samples range from a few hundred to hundreds of thousands. This is due to either time or monetary constraints. Running more simulations than this simply takes too long or costs too much. Based on these data I test my timing function on dimensions 3–8 and run up to 1,000,000 sample points.

4.3.3 Pipeline description

Despite the wide variety of application areas, all the simulation studies mentioned follow a standard procedure for analysis. They start with a uniform sampling of the parameter space. This is usually done with a space-filling design like a Latin hypercube [77], Halton sequence [43], or Centroidal Voronoi tessellation [29]. Without any knowledge of the relative importance of the dimensions they are sampled equally.

The simulation is run using each sample and the outputs are recorded. If the output is a complex object then feature extractors are run on the outputs to generate scalar results. Then, an emulator is built of this process. Prediction using this emulator must be fast as we will want to evaluate it at many points. Often, a Gaussian process model [92] is selected for the emulator.

With this emulator the user can now analyze the input/output relationship of the simulation. This can be done in a variety of ways, either by looking at static plots [22] or by interactively exploring the response surface [112]. We show an example of the HyperSlice technique for interactively exploring the response surface as implemented in Tuner [112] in Figure 4.4. In this case we show the conditional mean of the Gaussian process model. Tuner can also show the estimation variability if desired.

Interactive exploration of the Gaussian process model is a relatively new technique as it is more complex to implement and the limits in terms of the number of points and how the size of the kernels affects interactivity is not yet understood. The rest of this chapter will discuss how to address both these questions. I present a rendering algorithm which is similar to splatting [80] to render the Gaussian process prediction function using HyperSlice. I also develop a method to determine when the interactiv-

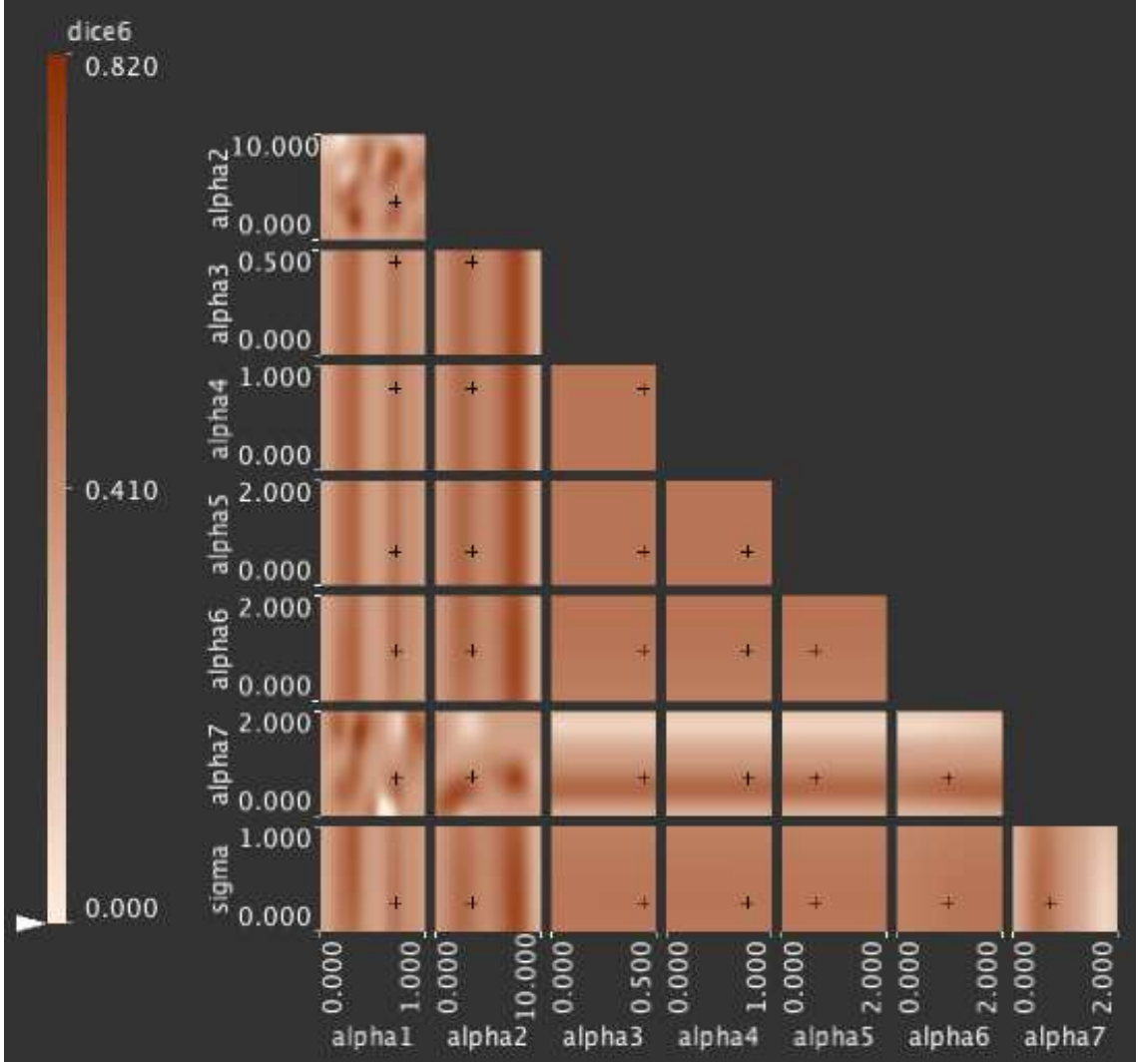


Figure 4.4: Screenshot of Tuner [112] demonstrating the HyperSlice [119] method for rendering an 8-dimensional parameter space using squared exponential kernel reconstruction on an image segmentation dataset. Here we show the view using the conditional mean of the Gaussian process model.

ity of the rendering will fail taking into account the geometric interpretation of the Gaussian process model as well as the performance of an individual user’s machine.

4.4 Requirements

Sampling the Gaussian process prediction formula Equation 4.3 directly in sufficient density to draw the slices is expensive because each evaluation requires computing the dot product of two size N vectors. Instead, I render the prediction formula in order to exploit the inherent parallelism provided by the GPU. Before describing the

algorithm itself, we first need to consider what the multi-dimensional “scene” we are trying to render looks like. The fundamental data type we are given are sample points of the simulation. This section describes how these sample points are transformed into higher-order geometric primitives and then what happens when slicing them in order to be drawn on screen.

4.4.1 Scene geometry

Here I will describe the spatial interpretation of the Gaussian process model so that one can build an intuition for the geometric portion of the prediction formula. This spatial interpretation is very close to the form used for the splatting algorithm.

The spatial interpretation of the Gaussian process model using squared exponential correlation is a set of multi-dimensional ellipsoids, one for each sample point. One may be tempted to think this is due to uncertainty at the sample points but this is not the case here as the outputs of a computer simulation are considered exact. The ellipsoids are due to giving the correlation functions compact support. In order to see why this is the case we first begin by looking at the formula for the “best linear unbiased prediction,” at an arbitrary location in the parameter space, x' , which is,

$$\hat{y}(x') = \mu + \vec{r}(x')R^{-1}(\vec{Y} - \mu)', \quad (4.3)$$

here \vec{r} is a vector of functions, one per sample point and each element of r , r_i , is the correlation between sample point x_i and x' . \vec{Y} is a vector of the sampled outputs. μ is the estimated process mean. R is the $N \times N$ correlation matrix between the sample points using the same correlation function. We also note that neither R^{-1} nor $(Y - \mu)$ depend on x' so let the vector, $\vec{\Upsilon} = R^{-1}(\vec{Y} - \mu)'$. Then, we can write Equation 4.3 in a linear form,

$$\begin{aligned} \hat{y}(x') &= \mu + r(x')\vec{\Upsilon} \\ &= \mu + \sum_{i=1}^N r_i(x')\Upsilon_i. \end{aligned}$$

The value Υ_i is $f'(x_i)$ and \hat{y} is \hat{f} from Equation 4.1 which is normalized by R^{-1} from Equation 4.3.

A common choice for $r(\cdot)$ is the squared exponential correlation function which has infinite support and is strictly positive meaning that it is defined everywhere in the domain and always returns some positive value however small. There is a different falloff parameter for each dimension. For visualization purposes these small values don't contribute a perceivable effect. Therefore, I set a lower bound on the correlation value which we denote, $\epsilon = 1 \times 10^{-9}$, essentially giving the correlation function compact support. This can also be done using specific compactly supported correlation functions as in Kaufman et al. [64]. The squared exponential correlation function is radial meaning the correlation amount between points decreases as the distance increases. This ϵ value essentially creates a d -dimensional ellipsoid region around each sample point with principal axis lengths related to the correlation falloff parameter. The sample point will only influence predictions within this region.

4.4.2 HyperSlice effect on scene geometry

The last step is using some method to examine this multi-dimensional scene on a computer screen. My chosen display method is the HyperSlice [119] technique to which will draw 2-dimensional slices through these multi-dimensional ellipsoids on screen. HyperSlice relies on the user selecting a *viewpoint* which determines the location in the multi-dimensional parameter space where to position of the slices. In Figure 4.5 I show the representation for the HyperSlice technique in both 2 and 3 dimensions. In 2D the slicing plane (what one sees) is a line. In 3D it is a 2D plane slicing through the space. In higher dimensions it is also a 2D slicing plane.

Only the points within range r of the slicing plane have an effect on the final image on the slice. The conceptual algorithm works by first filtering out all points that do not fall within range, r , of the slicing plane. Then, for all points within this range, I determine where the drawing plane intersects the ellipsoid which determines its impact on the drawing plane.

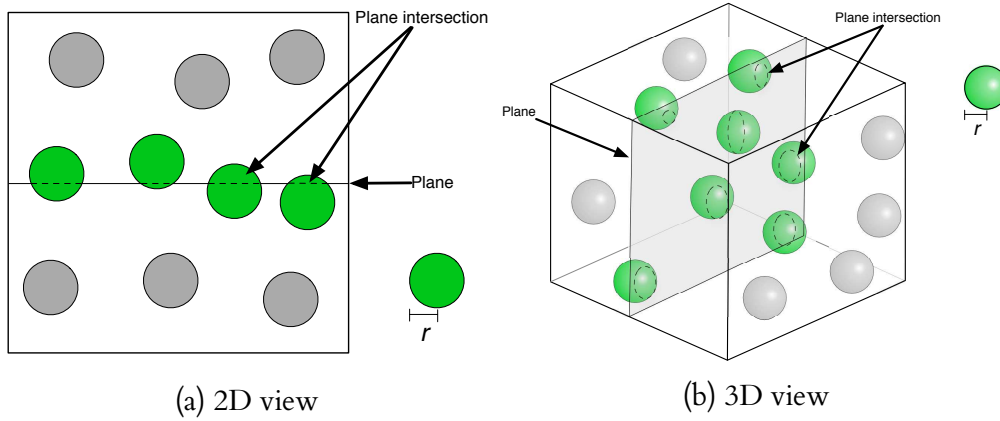


Figure 4.5: Diagrams showing how the HyperSlice [119] method “slices” through the kernels of the Gaussian process model in both 2D (a) and 3D (b). The slicing plane is denoted as “Plane” in the figure. This is the plane the user views. Only the points, denoted in green, fall within a distance r of the slicing plane will influence the rendered image. All other points can be filtered out as they do not affect the image. We then compute the influence of the unfiltered (green) points on the slicing plane.

4.4.3 Algorithm

The GPU has vertex and fragment processing stages. This is analogous to the filtering and rendering stages of our rendering algorithm. I present the full schematic of the rendering algorithm in Algorithm 3.

The distance to the slice is a $d - 2$ dimensional distance since the remaining 2 dimensions are projected on to the slice directly. I compute the projection of the point onto the current 2D slice in Algorithm 3 (Algorithm 3). I then compute the distance of the sample point to the slice in Algorithm 4. Because I am only interested in the distance to the slice itself, and not a particular point on that slice, I don’t include the two dimensions of the slice in the distance computation. If this distance is smaller than the size of the reconstruction kernel, I render a 2D slice through this reconstruction kernel. A speed-up for drawing exponential functions often used in GPU-based splatting algorithms is to use a template exponential distribution drawn onto a quad [80]. However, since these splat-like slices have different distances from each subplot they affect the subplot by different amounts depending on the distance. Therefore, I need to scale the intensity value of the texture by its distance to the slice (Algorithm 6).

In my vertex shader implementation I filter the points as well as compute the sliced

Algorithm 3 Rendering multi-dimensional data using HyperSlice and Gaussian process regression

Input: viewpoint \vec{v} , maximum distance r

Output: $\binom{d}{2}$ slices through an N -point, d -dimensional data set

for all $\binom{d}{2}$ subplots S_i **do** ▷ filtering

for all N points p in the vertex buffer **do**

$p_{2D} \leftarrow$ the 2D projection of p onto the slice S_i

$\text{dist} \leftarrow$ the distance of p to the slice S_i

if $\text{dist} < r$ **then** ▷ rendering

$\text{tex} \leftarrow$ the Gaussian splat scaled by dist

$\hat{p}_{2D} \leftarrow$ transform p_{2D} into screen coordinates

$w \leftarrow$ compute the splat width ($\sqrt{r^2 - \text{dist}^2}$)

 send 2D quads $(\hat{p}_{2D}(x) \pm w, \hat{p}_{2D}(y) \pm w)$ to the fragment shader to be

shaded with tex

end if

end for

end for

splat size. This splat size is used to generate a quad that I send to the fragment shader. I use the fragment shader to compute the final pixel color values within the slice. For practical visualizations this final pixel color value should be passed through a colormap. Therefore, in such practical applications, I recommend rendering the pixel values to a floating point texture. Then this floating point texture can be rendered to the screen with a colormap shader program to convert the floating point values to color values.

One of the main bottlenecks in the rendering pipeline is transferring vertex data from CPU memory to GPU memory. This is due to the slower speed of the bus compared to the GPU. I do not want the GPU waiting for pixel data. The best way to address this, as specified by Xue and Crawfis [125], is to store vertex data in display lists on the GPU. Then, before calling a draw command, I only need to update the small amount of viewpoint information to render the group of slices to the GPU. Hence, I store all N d -dimensional data points on the GPU. Memory of current GPUs is large enough that we can easily store millions of points in a number of dimensions directly on the card.

4.5 Derivation of scene geometry

We now turn to a formulation for the expected total running time to draw N points in d dimensions within a slice distance of r using our algorithm above. My complexity analysis is based on the fact that my rendering algorithm can be decomposed into a pipeline with filtering and drawing steps. I also assume, as exemplified in subsection 4.3.2, that my points are uniformly distributed in data space. My mathematical derivation also assumes that the ellipsoids generated by the Gaussian process model are, in fact, hyperballs. While this may seem like an over-simplification, the principal axes of the ellipsoids are axis-aligned with respect to the parameter space (see subsection 4.4.1). An ellipse is simply “stretching” the parameter space by a fixed amount in each direction.

The two stages of rendering mean that the measured time is the time to run the filtering stage plus the time to run the drawing stage. However, because of the pipeline setup of the GPU, a low number of fragments can be drawn “for free” on the spare compute capacity of the card not being used for filtering. Once this spare capacity is exhausted, the rendering time will dominate the total drawing time. Therefore, the total drawing time, t_{total} , is the time to filter the points plus the time to render the points on screen but only after a certain number of fragments are drawn. I represent this breaking point with $I(\text{frags} > a)$ which is an indicator function that returns 0 when the number of fragments is less than the break-point, a , and 1 otherwise.

$$t_{\text{total}} = t_{\text{filter}} + I(\text{frags} > a) * t_{\text{render}} \quad (4.4)$$

4.5.1 Filtering

In the filtering step (lines Algorithm 4 and Algorithm 5 of Algorithm 3) I take each data point and compute its distance to each plot in order to determine if it is worth the effort to actually draw the quad. For each sample point and for each slice, I compute the distance from the sample point to the slice. If the distance is less than r I draw it.

Since there are subplots for each pair of dimensions there are $\binom{d}{2}$ subplots in total. Filtering is a constant time per point but is architecture-dependent. I denote this time t_f and the total filtering time, t_{filter} , is

$$t_{\text{filter}} = t_f \binom{d}{2} N. \quad (4.5)$$

4.5.2 Rendering

During the rendering step, the algorithm only needs to process a fraction of the N points that are visible. I call this fraction N' . The rest of the points are discarded in the filtering code on the GPU. In the case of HyperSlice, the rendering time is significant. Besides having to determine the size of the quad to be rendered in lines Algorithm 7 and Algorithm 8 of Algorithm 3, the actual rendering time depends on the number of pixels covered by the quad since each pixel requires a constant time to draw. Because of this, my formulation for the rendering time must include the quad size for each point rendered, q_i , and the time to render each pixel in a quad, t_H ,

$$t_{\text{render}} = t_H \sum_{i=1}^{N'} q_i. \quad (4.6)$$

4.5.3 Expected total time

Equation 4.4 gives the total running time for a particular configuration of N sample points in d dimensions and for a particular viewpoint \vec{v} . However, I am interested in how well the rendering algorithm performs under many different configurations of points and viewpoints. The worst case performance is when the full kernel needs to be drawn. In other words, when the kernels are not cut off by the edges of the parameter space and the view point is in the center. However, it is important to know how the rendering will perform as the user views a set of different plots. Hence, a much more useful measurement is the *average* time to draw the view over all possible point configurations and all possible viewpoints. In order to compute this, the expected rendering time, $E[t_{\text{total}}]$, is an average over all point configurations and viewpoints in the unit cube $[0, 1]^d$:

$$E[t_{\text{total}}] = t_f \binom{d}{2} N + I(\text{frags} > a) t_H E[N'] E[q], \quad (4.7)$$

Here, the first term represents the time to filter all N points over the $\binom{d}{2}$ plots and the second term is the time to draw any points that pass the filter.

The total amount of rendering that needs to be done is the number of points passing the filtering stage times the size of these points on the slice. The quantity t_H is the time to draw a single fragment using the HyperSlice method, $E[N']$ is the expected number of points within a distance of r from all 2D slices of the subplot matrix, and $E[q]$ is the expected size of a quad drawn.

$E[N']$ is based on the total number of sample points we need to process times the expected percentage of points that will be within range of the slices. There are N points to process for each of the $\binom{d}{2}$ subplots. For a single 2D slice, denote the expected percentage of points within a distance r in d dimensions $\widehat{N'}(r, d)$. The percentage of points can be expanded into

$$\begin{aligned} E[N'] &= \binom{d}{2} N \cdot \widehat{N'}(r, d) \\ &= \binom{d}{2} N \cdot \sum_{i=0}^{d-2} (-1)^i \binom{d-2}{i} \frac{\pi^{d-2-i} r^{d-2+i}}{\Gamma\left(\frac{d-2+i}{2} + 1\right)}. \end{aligned} \quad (4.8)$$

$\widehat{N'}(r, d)$ is the sum of higher and higher dimensional spheres as they are sliced by 2-dimensional planes. For the full derivation, please see chapter B.

For the HyperSlice technique, the quantity $E[q]$ depends on the size of the spherical reconstruction kernel which depends on r and d . Denote this quantity $\widehat{Q}(r, d)$. This represents the expected number of fragments produced on a particular slice when the sample point is within a distance r of the slice. As described in chapter C of the

appendix, $E[q]$ expands into,

$$\begin{aligned}
E[q] &= \widehat{Q}(r, d) \\
&= \frac{1}{\widehat{N}'(r, d)} \sum_{i=0}^{d-2} (-1)^i \binom{d-2}{i} [\text{corner}(d, i, r) \\
&\quad - \text{side}(d, i, r) \\
&\quad + \text{center}(d, i, r)] \\
&= \frac{1}{\widehat{N}'(r, d)} \sum_{i=0}^{d-2} (-1)^i \binom{d-2}{i} \left[\frac{4\pi^{(d-i)/2-1} r^{d+i}}{\Gamma((d+i)/2 + 1)} \right. \\
&\quad - \frac{3\pi^{(d-i-1)/2} r^{d+i+1}}{\Gamma((d+i+3)/2)} \\
&\quad \left. + \frac{2\pi^{(d-i)/2-1} r^{d+i+2}}{\Gamma((d+i)/2 + 2)} \right]. \tag{4.9}
\end{aligned}$$

Here the $\text{corner}(d, i, r)$, $\text{side}(d, i, r)$, and $\text{center}(d, i, r)$ functions correspond to derivations 1, 2, and 3 listed in subsection B.3.2 of the appendix respectively. While the current formula appears quite complex, it is fast and easy to evaluate on a computer. In fact without this formula the computations would be intractable. There might exist a more comprehensible formula, however, this is beyond the scope of this paper and is a subject for future work.

These formulas take into account the size of the kernel even if part of it is clipped by the edge of the parameter space. This is very important for larger kernels in higher dimensions. There, the volume of a kernel is very small but the radius may be very large and therefore the kernel is *always* clipped by the edges of the parameter space. These corner and edge terms will dominate in higher dimensional cases or for large values of r . In lower dimensional cases, or smaller values of r the center term will contribute more to the final rendering time.

4.6 Fitting

With a proper mathematical formulation of the scene geometry in terms of how many pixels are produced on screen, I now describe the second missing piece of the prediction, namely, a procedure for tuning this formula to a particular user's implementation

and hardware configuration. The values t_f and t_H in Equation 4.7 are dependent on a particular hardware. Hence, I engage in an empirical stage to determine these values for a particular rendering environment. For the purposes of these timings I set my correlation cutoff value (see subsection 4.4.1), ϵ , to 1×10^{-9} . This value allows me to link the kernel radius, r , with the kernel bandwidth parameter in the GP model.

The architecture of each GPU is different and what is efficient on one architecture may not carry over to another. Therefore, it is impossible to argue from first-principles how to derive t_f and t_H for a particular GPU. I fit these parameters by doing a regression analysis on empirical results obtained from examining the time to render a frame for various values of d , N , and r . While the specific values I derive are specific to a particular architecture, the method we present is applicable elsewhere.

In order to fit Equation 4.7 I note that the first term represents the number of points we need to check to be filtered. This is constant with respect to the kernel size, r . The second term, the drawing time, increases with respect to r . The filtering time dominates for small values of r while the drawing time dominates for larger values of r . Therefore, there will be a point in terms of number of fragments drawn, designated a , at which point the dominant term will change from the first to the second. In order to fit this behavior I used a segmented regression model which changes behavior according to the value of a $\{0,1\}$ indicator function, $I(\text{frags} < a)$, where frags is the total number of fragments drawn. This function returns 0 if $\text{frags} < a$. I form the regression formula as:

$$t_{\text{render}} = N \binom{d}{2} t_f + I(\text{frags} > a) t_H \text{frags}. \quad (4.10)$$

This formula contains three parameters to be estimated: t_f (the time to filter one sample point), t_H (the time to render one fragment), and a (the crossover point).

4.6.1 Sampling

For each dimension, I must ensure that I have good coverage of the number of fragments being drawn. Hence, I must choose different values of r for each d . In order to obtain a sensible range of values for $\widehat{Q}(r, d)$ we begin by using dimension 3 as a

baseline and vary radii from 0 to 0.5 in that dimension to come up with a reasonable range of $\hat{Q}(r, d)$. Given this desired range of fragments, for the remaining dimensions I numerically invert Equation 4.9, given d and hence, obtain a range of radii r .

The final issue is that for each setting of d , N , and r I must generate enough iterations such that the average number of points affecting the slices, N' , converges to the theoretical expected value, $E[N']$. This is the expected number of points need to be drawn over all possible uniform distributions of points and viewpoints. In my case we found that 20 viewpoint changes, redrawing the screen for a fixed viewpoint 5 times, and 3 sample point distributions for a total of 300 timing measures resulted in good convergence.

I compute the number of fragments drawn on screen internally by the application. I can do this because we know the position of the focus point, locations of all the sample points, and the kernel information. The number of fragments in the calculations is the percentage the quad takes up on the subplot. In other words, if a quad takes up an entire subplot then it has area 1. This measurement serves as a proxy for the number of fragments generated in the GPU. OpenGL offers a query object, `GL_SAMPLES_PASSED`, that should return the number of fragments needed to draw the screen. This is very convenient and would correctly account for the rasterization method used. However, in my experiments this query would return several different values for the exact same scene, which does not make sense. Due to this inconsistency, I used an internal calculation.

I can record the rendering time with either the CPU timer or the GPU timer. The CPU timer better represents the user's perception of how long it takes to draw the screen since it includes the time necessary to transfer data back and forth to the GPU. However, this timer is much noisier than the GPU timer. In my tests I found that on average the CPU timing differed from the GPU timing by a constant amount. Therefore, I used the GPU timer for the timing. The GPU timer is still quite noisy however and in order to smooth out this noise I redrew each screen five times for each change of viewpoint and then averaged the times.

4.6.2 Final model

If we then apply these estimates of filter and draw time to Equation 4.7, the full form of the prediction model, conditional on the dimension, d , is

$$\begin{aligned}
 t_{total}(d, N, r) &= t_f(d) \binom{d}{2} N + I(\text{frags} > a) t_H(d) E[N'] E[q] \\
 &= t_f(d) \binom{d}{2} N + I(\text{frags} > a) t_H(d) \binom{d}{2} N \widehat{N}'(r, d) \widehat{Q}(r, d) \\
 t_{total}(d, N, r) &= \binom{d}{2} N \\
 &\quad \left[t_f(d) + I(\widehat{N}'(r, d) \widehat{Q}(r, d) > a(d)) t_H(d) \widehat{N}'(r, d) \widehat{Q}(r, d) \right], \quad (4.11)
 \end{aligned}$$

where $t_f(d)$, $t_H(d)$, and $a(d)$ are the parameters fit and $\widehat{N}'(r, d)$ and $\widehat{Q}(r, d)$ are from Equation 4.8 and Equation 4.9 respectively. The parameters are conditional on the dimension, d , because I fit each dimension separately so there is a different value of t_f , t_H , and a for each dimension. I fit these parameters using segmented regression as described in subsection 4.7.1.

4.7 Timing results

I now present the results of running the timing experiments. My test machine is a Macbook Pro with Retina display with a 2.6GHz Intel Core i7, 16GB of RAM, and an NVIDIA GeForce GT 650M graphics card with 1GB of graphics memory. In order to produce consistent results I disabled the GPU power management extension. With it enabled the system varies the clock speed of the GPU while the experiments are running, producing inconsistent results.

4.7.1 Data fitting

I plot the rendering time as a function of number of fragments, drawn in Figure 4.6 for different values of N , d , and r . Each dimension is treated separately as my estimation procedure is volume-based and volumes are not readily comparable between dimensions. In particular, the units of volume depend on the dimensionality and the

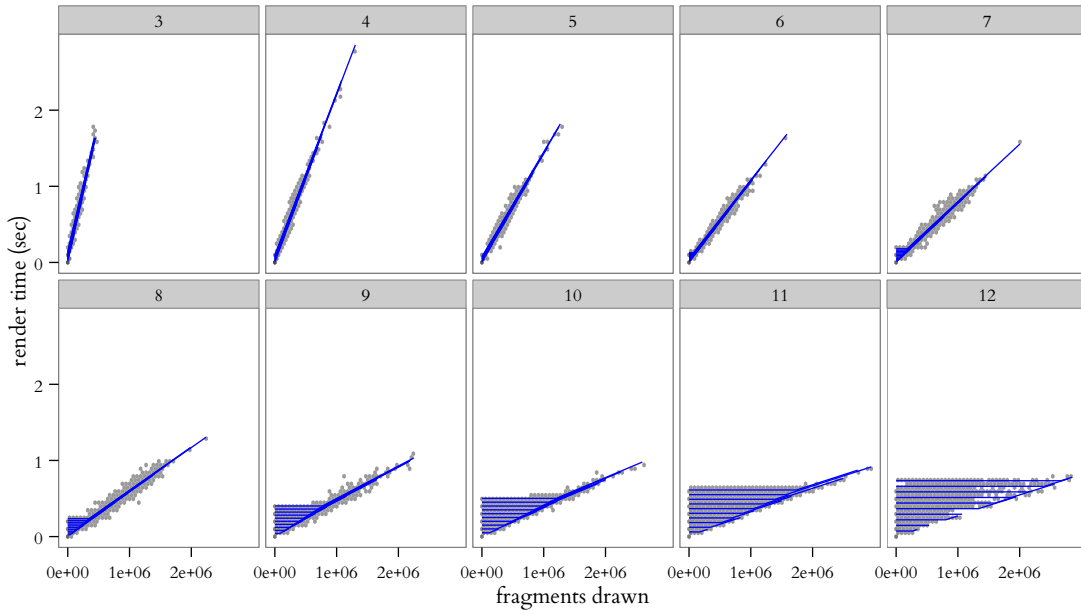


Figure 4.6: Scatterplots of the time to render using the HyperSlice method. Each dimension is analyzed separately. The x-axis is the number of fragments drawn on screen and the y-axis is the number of seconds recorded by the GPU timer for the frame to draw. The blue line is the predicted rendering time using my fitted formula.

relationship between radius and volume, for example, a 3-dimensional ball is very different from a 5-dimensional ball. Furthermore, the dimensionality of the data is usually given and not variable while one can vary the number of sample points. The x-axis in the figure is the actual number of fragments drawn on screen and the y-axis is the time, in seconds, to draw the frame. As predicted by Equation 4.7, the rendering time remains constant while the GPU is primarily filtering points and then increases linearly with the number of fragments once the drawing stage dominates.

To fit these data I first computed the fragments and rendering time per sample by dividing the recorded fragments and time by $N\binom{d}{2}$ since our prediction function, Equation 4.10, is linear in $N\binom{d}{2}$. I also filtered out any experiments where the rendering time was greater than 1 second since this would extend the sampling time and I am primarily concerned with finding interactive times. I then fit a basic linear model and a 2-segment regression model using the segmented package [81] in R. If the slope of the two segments did not differ significantly then I simply used the linear model and set the break-point to 0. For these dimensions the rendering time always dominates. I found that if the ratio between slopes of the 2-segment regression was greater than

10 then I got a better fit with the 2-segment regression than with a single linear fit.

The blue line shown in Figure 4.6 is the predicted rendering time versus the number of fragments drawn. Here the blue line goes directly through the cloud of timing points. The multiple horizontal lines within each dimension correspond to the different values of N we used in our experiments. We can also see as the dimensionality increases, the filtering time begins to dominate. This is because for each subplot of the HyperSlice the algorithm must filter all N points and the number of subplots increases as $O(d^2)$. We can also see how the slope of the rendering time line (t_H in Equation 4.10 and Table 4.2) decreases as the dimensions increase. This is because the screen size for running the experiments is fixed so as the number of dimensions increase the area of each individual plot becomes smaller since we have $\binom{d}{2}$ HyperSlices. Therefore, in higher dimensions we have fewer pixels to process.

The table of parameters by dimension for my reference system is listed in Table 4.2. Here the relationship between the number of dimensions and the fitted parameters is more apparent. For lower numbers of dimensions (3–5), it is difficult to directly measure the filtering time (t_f) as the rendering time always dominates. In this case t_f is just the y-intercept of the fit line for the rendering time. For higher values of d ($d > 5$), one can directly measure the filtering time. The reason t_f increases between dimensions 8 and 9 is because in the filtering code I parallelize the distance computations in OpenGL using the `vec4` type for every group of dimensions. So, an additional group of `vec4`s is required for dimensions 9–12 and computing distances with these additional `vec4`s takes slightly more time.

4.7.2 Accuracy

As with the filtered scatterplot, I compared our predicted running time against new timing data using the same experimental conditions. I do this in order to test new values of r . I then compared the predicted rendering time against the empirically recorded ones. Figure 4.7 shows the absolute and relative errors for prediction using the HyperSlice method and squared exponential kernel regression.

Table 4.2: Table showing the calibrated parameters, a , t_f , and t_H , as a result of running a segmented regression fit using the data I gathered from my timing experiments and Equation 4.11. The factors t_f and t_H are in nanoseconds and a is defined in terms of fragments per sample.

d	a	t_f (ns)	t_H (ns)
3	0.000	48.100	3470
4	0.000	15.600	2130
5	0.000	7.430	1380
6	0.00504	8.550	1030
7	0.00859	8.560	755
8	0.0123	8.400	569
9	0.0214	11.200	427
10	0.0267	11.200	342
11	0.0316	11.100	263
12	0.0404	11.100	255

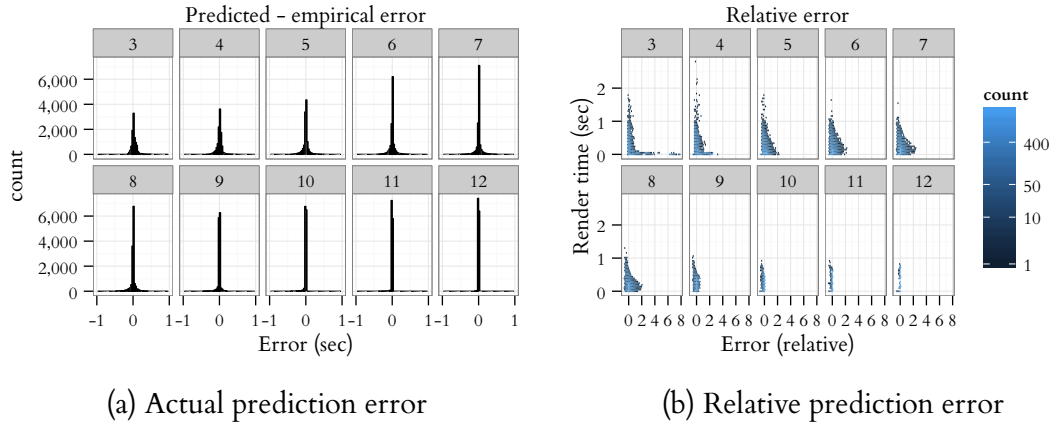


Figure 4.7: Histograms showing actual (a) and relative (b) error rates for the Hyper-Slice method comparing predictions using Equation 4.10 to empirical results. I show (b) as a scatterplot in order to demonstrate that the largest relative errors occur when the drawing times are smallest. Each dimension is treated separately since the units of volume differ for each dimension and I have computed the filtering time, t_f , and drawing time, t_H , separately for each dimension.

Many of the largest relative errors occur for the smallest total rendering times so *any* miscalculation will result in a large relative error. The actual difference, is shown as a histogram in Figure 4.7a. Each sub-plot is a separate dimension. I show the difference between the predicted and measured rendering times, in seconds, on the x-axis. Every dimension has a strong spike at 0 indicating that most of the predictions are off by a very small amount with only a few being very inaccurate. I also show

the relative error as a hexagonal-binned plot in Figure 4.7b. A hexagonal-binned plot [17] is a 2d density plot using a hexagonal grid as the binning primitive. The x-axis is the percent error between the predicted time and the measured time relative to the recorded time and the y-axis is the measured time to render the frame. Many of the rendering times are very small so any error in the prediction results in a very high relative error.

In order to check the predictive ability of our procedure we also performed a 5-fold cross validation. For each dimension, we split the data so that 20% is used for building the model and the remaining 80% used for testing. We then use the testing set to compute the difference between the predicted and expected rendering times. We then repeat this procedure four more times using the next 20% partition for training. We then compute the root-mean squared error and maximum absolute error between the prediction and the recorded values. The results are shown in Table 4.3. While the relative errors may seem high these occur when predicting very small times so any error will be high on a relative basis. I also show the Nash-Sutcliffe efficiency [85] for each dimension, which is the ratio of variance explained by our model to the total variance. This ranges from $-\infty$ to 1 where values close to 1 mean that the model explains most of the total variance. A value over 0 is considered an acceptable level of performance [79]. All of the values in Table 4.3 are very close to one so my model contains a great deal of information from the data.

4.8 Application scenarios

As was mentioned in section 3.1, there are a number of ways to apply our prediction methodology in a practical visualization system. To this end I show two application scenarios where my method can be used to control the number of samples to maintain interactive rates. I show how our method may be used to sample the simulation in order to maintain a desired frame rate and to subsample an existing dataset in order to attain interactive frame rates.

Table 4.3: Results of the cross-validation procedure. For each dimension I compute the root-mean squared error of prediction as well as the Nash-Sutcliffe efficiency [85], and the relative maximum error. The Nash-Sutcliffe efficiency is the ratio of the variance explained by our prediction model to the total variance. All errors are in terms of seconds.

d	RMSE	Nash-Sutcliffe	Relative max error
3	0.0685	0.931	0.624
4	0.0669	0.922	0.703
5	0.0702	0.888	0.403
6	0.0735	0.848	1.340
7	0.0726	0.801	0.383
8	0.0661	0.791	0.556
9	0.0398	0.920	0.601
10	0.0246	0.977	0.461
11	0.0127	0.996	0.386
12	0.00456	1.000	0.233

4.8.1 Constrain sampling

Figure 4.8 is a dialog box for the Tuner [112] system. The task is to enter the number of sample points to take from the simulation. The dialog is driven by Equation 4.10. When the user changes the number of samples directly (a), the dialog computes the expected frame rate and displays that to the user in (b). As an alternative method, the user may value interactivity highly and consequently selects the number of sample points to take by entering the desired frame rate (b) and letting the system select the number of samples.

4.8.2 Subsample points

The goal of this algorithm, presented as Algorithm 4 is to reduce the sample size, N , such that the rendering time reaches an acceptable 30fps. Normally this is done by removing samples from the set. An issue with simply removing points and rebuilding the Gaussian process model is that the bandwidth parameters, θ will change.

In Figure 4.9 I show the trade-off between the radius, r , and the number of points, N , that can be drawn in 30fps. When subsampling data I expect that the radius around

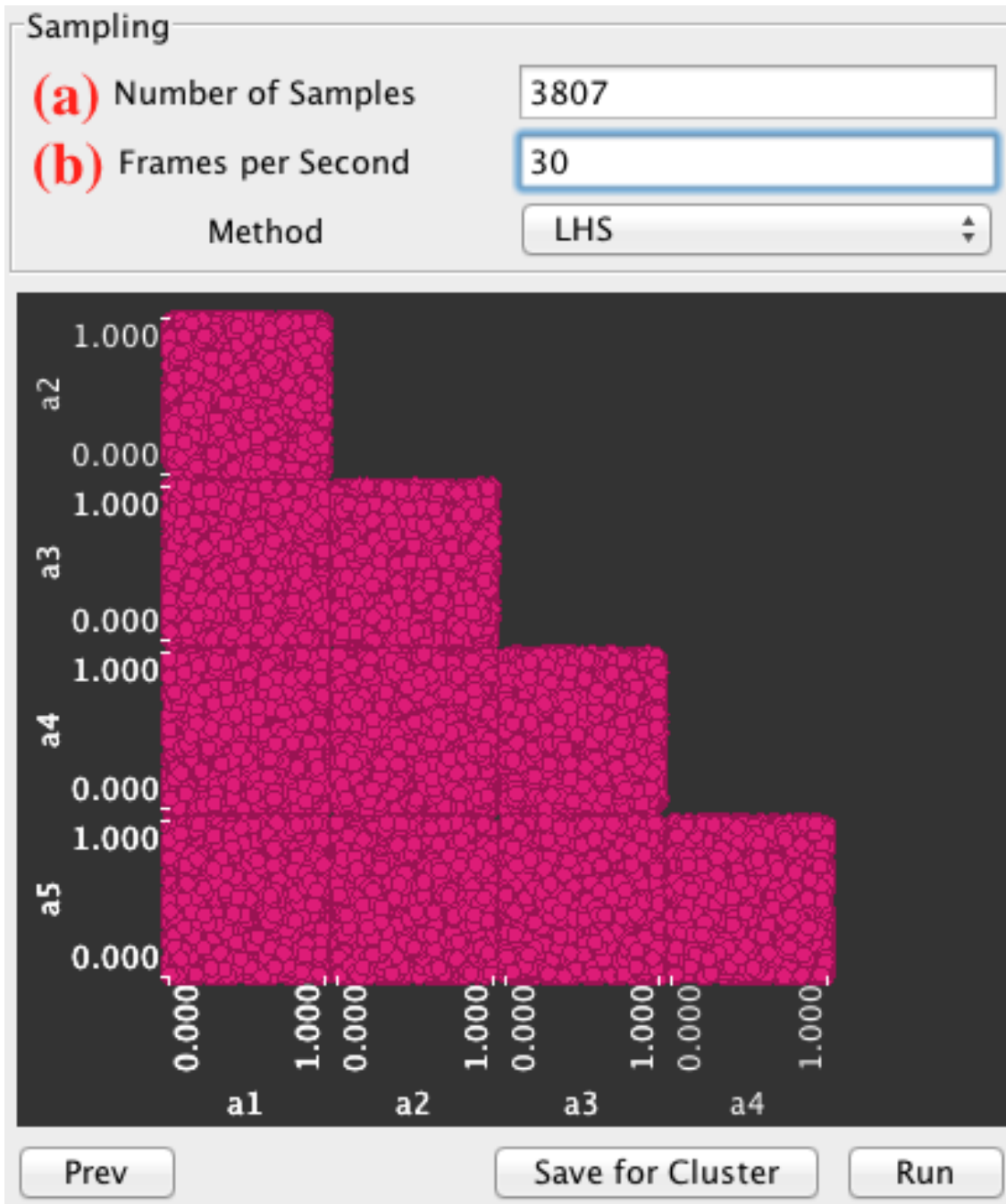


Figure 4.8: A prototypical example use case for my prediction formula, Equation 4.10. The user is able to either enter the number of sample points directly in field (a) and the system displays the expected fps in (b) or enter the desired fps in (b) first and the system calculates the number of sample points.

each sample point increases as the number of sample points decreases. The goal of Algorithm 4 is to lower the number of sample points until this line is reached.

In this fashion one can have a progressive rendering setup using 2 GP models, a low-resolution model for fast rendering and a high-resolution one for detail views. The system could dynamically switch between these two when interacting.

Algorithm 4 A proposed algorithm for subsampling data in order to achieve interactive rendering times using the Gaussian process model with the HyperSlice rendering technique.

Input: Calibrated prediction formula $E[t_{\text{total}}^H](N, d, r)$, calibrated GP model parameters $\vec{\theta}$

$t_{\text{pred}} \leftarrow E[t_{\text{total}}^H](N, d, r)$

while $t_{\text{pred}} < 30\text{fps}$ **do**

$N_{30\text{fps}} \leftarrow$ Numerically solve $E[t_{\text{total}}^H](N, d, r)$ for an N that will give 30fps rendering times

 Uniformly remove $N - N_{30\text{fps}}$ sample points

$r' \leftarrow$ Rebuild the GP model, thereby recomputing r

$t_{\text{pred}} \leftarrow E[t_{\text{total}}^H](N_{30\text{fps}}, d, r')$

end while

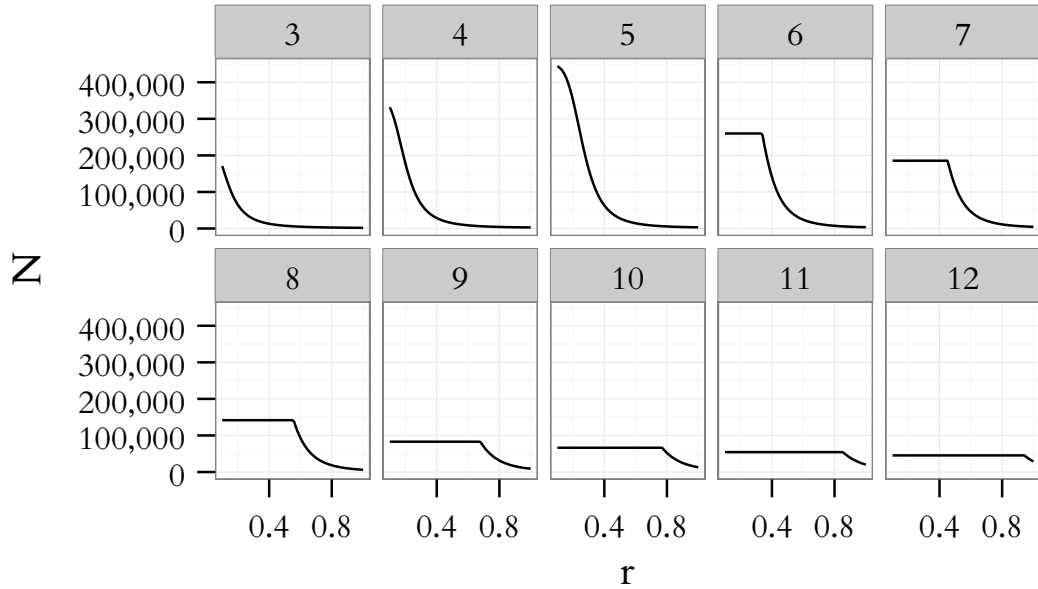


Figure 4.9: Average number of points that can be rendered in 30 frames per second for the HyperSlice technique.

4.9 Limitations and future work

In this chapter I have presented the characteristics of data used for analyzing computer simulations under the *design and analysis of computer experiments* framework [97]. I investigated how interactive rendering times may be used in this framework using the HyperSlice [119] rendering technique implemented on the GPU. I then describe a method using both the scene geometry and estimates of the user's machine capabilities in order to make an accurate prediction of the rendering time. I find that timing,

especially using wall-clock time, is extremely noisy and makes fitting very difficult. It is much more reliable to use the timer on the GPU itself, however one must take into account the time needed to return back from the GPU which in our experiments is about 1ms.

In the future, I will investigate how to reduce the number of trials needed to properly fit the formula. Currently each dimension takes about 6 hours to complete and requires the machine to be dedicated to running the timing code. Reducing the number of trials will help to alleviate this time consuming task.

It would also be interesting to extend the approach to the analysis of HyperSlice rendering in general. The basic geometrical operation in our mathematical model is slicing multi-dimensional spheres with 2D planes and estimating the area. Therefore our method should work directly with any radial basis function reconstruction technique like the work by Hong et al. [53] although I have not directly tested this. I would also like to extend my mathematical model to take the shape of the reconstruction primitive into account. This would allow the analysis of the timing of HyperSlice rendering using a much more broad set of reconstruction methods like nearest neighbor or linear regression. The framework can also be used to estimate the rendering time of density estimation by setting $f(x_i) = 1$ in Equation 4.1. Repeating the analysis using box primitives would allow me to estimate the time complexity of some of the recent real-time large-data aggregation and visualization methods like imMens [73] and NanoCubes [72]. Both these methods use rectangular binning in their density estimation.

I will also implement our subsampling strategy. There is a lot of overdraw occurring which does not contribute at all to the final plot as the color channel is effectively maxed out, especially as the value of N and r increases. By detecting when this occurs and only rendering the first few points we could improve rendering efficiency.

5 | Conclusion

Understanding multi-dimensional spaces is difficult. Visualization can give us context to help understand the geometry. With the direct visualization of these multi-dimensional continuous datasets through slice views, we can use a familiar concept to give context and meaning to a complex task.

Multi-dimensional continuous functions are commonly visualized with 2D slices or topological views. With Sliceplorer, I explore 1D slices as an alternative approach to show such functions. My goal with 1D slices is to combine the benefits of topological views, that is, screen space efficiency, with those of slices, that is a close resemblance of the underlying function. I compare 1D slices to 2D slices and topological views, first, by looking at their performance with respect to common function analysis tasks. I also demonstrate 3 usage scenarios: the 2D sinc function, neural network regression, and optimization traces. Based on this evaluation, I characterize the advantages and drawbacks of each of these approaches, and show how interaction can be used to overcome some of the shortcomings.

I also presented Hypersliceplorer, an algorithm for generating 2D slices of multi-dimensional shapes defined by a simplicial mesh. Often, slices are generated by using a parametric form and then constraining parameters to view the slice. In this case, I developed an algorithm to slice a simplicial mesh of any number of dimensions with a two-dimensional slice. In order to get a global appreciation of the multi-dimensional object, I show multiple slices by sampling a number of different slicing points and projecting the slices into a single view per dimension pair. These slices are shown in an interactive viewer which can switch between a global view (all slices) and a local view (single slice). I show how this method can be used to study regular polytopes,

differences between spaces of polynomials, and multi-objective optimization surfaces.

Finally, I develop a method for predicting the rendering time to display multi-dimensional data for the analysis of computer simulations using the HyperSlice [119] method with Gaussian process model reconstruction. My method relies on a theoretical understanding of how the data points are drawn on slices and then fits the formula to a user's machine using practical experiments. I also describe the typical characteristics of data when analyzing deterministic computer simulations as described by the statistics community. I then show the advantage of carefully considering how many data points can be drawn in real time by proposing two approaches of how this predictive formula can be used in a real-world system.

5.1 Future

My work has had a major focus on using direct visualization techniques to understand multi-dimensional continuous spaces. My intention is that this work can be expanded upon to herald in a new era of multi-dimensional data analysis. In my opinion, the major innovations preventing this technique from being used in a broader application are a library for slicing multi-dimensional spaces and more user-focused projects. Building on these two thrusts will move multi-dimensional continuous data analysis to the mainstream.

One of the reasons for the lack of adoption for slice-based visualization of multi-dimensional objects is the complete lack of software to generate even static slice views. There are many libraries for popular data analysis languages like Python, Javascript, and R. In order to make slice based views more viable I plan to develop an interactive slice-based visualization software based on the prototype tools I have already developed. This will lower the cost of entry of slice-based views of multi-dimensional continuous datasets. The end result is more users familiar with this visualization type.

In addition, more focused projects with end-users in the form of design studies [99] will help to develop both the task taxonomy and the visualization techniques. As part of the task abstraction, we can learn how these users' tasks fit in with the task and data

taxonomy proposed in this thesis. Then we can refine and extend the task and data taxonomy. This taxonomy will allow visualization researchers to identify gaps and develop tools to address them, thus creating more effective visualizations of multi-dimensional continuous data.

5.2 Implications

The main goal of my thesis was to explore what is possible with slice-based visualizations of continuous multi-dimensional datasets. My hope is that this work will serve as a basis for an increasing focus on direct visualization of multi-dimensional objects. Often it seems that the default analysis technique for more than three dimensions is to reduce the dimensionality of the data and then render the reduced data on screen. This suffers from issues of distortion of distances and relative sizes. The analysis tasks for multi-dimensional data are all developed around understanding the carefully chosen dimensions. Hence, transforming these dimensions takes away a lot of contextual knowledge about the simulation.

I also hope to bring more attention to continuous multi-dimensional data analysis. In the visualization community, most of the work on multi-dimensional and high-dimensional data has focused on the discrete case. There are many task taxonomies, techniques, and applications for discrete data. My hope with this thesis is that by developing a task and data taxonomy as well as an in-depth study of direct visualization techniques will bring similar attention to multi-dimensional continuous data analysis. There are a number of under-explored application areas in this field. I have identified some in my own work, but with further research in this field will bring more knowledge and understanding about how we, as three-dimensional beings can understand multi-dimensional continuous datasets.

Bibliography

- [1] David H. Ackley. *A Connectionist Machine for Genetic Hillclimbing*. The Kluwer International Series in Engineering and Computer Science. Springer, 1987.
- [2] Robert Amar, James Eagan, and John T. Stasko. Low-level components of analytic activity in information visualization. In *IEEE Symposium on Information Visualization (InfoVis)*, pages 111–117. IEEE Computer Society, October 2005.
- [3] Robert Amar and John Stasko. A knowledge task-based framework for design and evaluation of information visualizations. In *IEEE Symposium on Information Visualization*. Institute of Electrical & Electronics Engineers (IEEE), October 2004.
- [4] Daniel Archambault, Helen Purchase, and Bruno Pinaud. Animation, small multiples, and the effect of mental map preservation in dynamic graphs. *IEEE Transactions on Visualization and Computer Graphics*, 17(4):539–552, April 2011.
- [5] Daniel Asimov. The grand tour: A tool for viewing multidimensional data. *SIAM Journal on Scientific and Statistical Computing*, 6(1):128, January 1985.
- [6] Sven Bachthaler and Daniel Weiskopf. Continuous scatterplots. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1428–1435, December 2008.
- [7] Thomas Bäck. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, USA, 1996.

- [8] C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. The quick-hull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 22(4):469–483, December 1996.
- [9] Giuseppe Di Battista and Roberto Tamassia. Algorithms for plane representations of acyclic digraphs. *Theoretical Computer Science*, 61(2):175–198, November 1988.
- [10] Wolfgang Berger, Harald Piringer, Peter Filzmoser, and Eduard Gröller. Uncertainty-aware exploration of continuous parameter spaces using multivariate prediction. *Computer Graphics Forum*, 30(3):911–920, June 2011.
- [11] Steven Bergner, Michael Sedlmair, Torsten Möller, Sareh Nabi Abdolyousefi, and Ahmed Saad. ParaGlide: Interactive parameter space partitioning for computer simulations. *IEEE Transactions on Visualization and Computer Graphics*, 19(9):1499–1512, September 2013.
- [12] Jacques Bertin. *Sémiologie Graphique: Les diagrammes, les réseaux, les cartes*. Gauthier-Villars, 1967. (Translation 1983. *Semiology of graphics* by William J. Burg).
- [13] Praveen Bhaniramka, Rephael Wenger, and Roger Crawfis. Isosurfacing in higher dimensions. In *Proceedings of the conference on Visualization '00*, pages 267–273. IEEE Computer Society, IEEE, 2000.
- [14] Stephan Borzsony, Donald Kossmann, and Konrad Stocker. The skyline operator. In *Proceedings 17th International Conference on Data Engineering*. IEEE Computer Society, 2001.
- [15] George E. P. Box and Norman R. Draper. *Response Surfaces, Mixtures, and Ridge Analyses*. Wiley Publishing, Hoboken, NJ, USA, 2nd edition, April 2007.
- [16] Matthew Brehmer and Tamara Munzner. A multi-level typology of abstract visualization tasks. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2376–2385, December 2013.

-
- [17] Daniel B. Carr, Richard J. Littlefield, Wesley L. Nicholson, and J. S. Littlefield. Scatterplot matrix techniques for large N. *Journal of the American Statistical Association*, 82(398):424–436, June 1987.
 - [18] Hamish Carr and David Duke. Joint contour nets. *IEEE Transactions on Visualization and Computer Graphics*, 20(8):1100–1113, August 2014.
 - [19] Hamish Carr, Zhao Geng, Julien Tierny, Amit Chattopadhyay, and Aaron Knoll. Fiber surfaces: Generalizing isosurfaces to bivariate data. *Computer Graphics Forum*, 34(3):241–250, June 2015.
 - [20] Hamish Carr and Jack Snoeyink. Path seeds and flexible isosurfaces using topology for exploratory visualization. In *Proceedings of the symposium on Data visualisation 2003*, pages 49–58. Eurographics Association, May 2003.
 - [21] Hamish Carr, Jack Snoeyink, and Ulrike Axen. Computing contour trees in all dimensions. *Computational Geometry*, 24(2):75–94, February 2003.
 - [22] William L. Chapman, William J. Welch, Kenneth P. Bowman, Jerome Sacks, and John E. Walsh. Arctic sea ice variability: Model sensitivities and a multidecadal simulation. *Journal of Geophysical Research: Oceans*, 99(C1):919–935, January 1994.
 - [23] William S. Cleveland and Robert McGill. Graphical perception: Theory, experimentation, and application to the development of graphical methods. *Journal of the American Statistical Association*, 79(387):531–554, 1984.
 - [24] Carlos D. Correa, Peter Lindstrom, and Peer-Timo Bremer. Topological spines: A structure-preserving visual representation of scalar fields. *IEEE Transactions on Visualization and Computer Graphics*, 17(6):1842–1851, August 2011.
 - [25] Ivo Couckuyt, Frederick Declercq, Tom Dhaene, Hendrik Rogier, and Luc Knockaert. Surrogate-based infill optimization applied to electromagnetic problems. *International Journal of RF and Microwave Computer-Aided Engineering*, 20(5):492–501, September 2010.

- [26] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, April 2002.
- [27] Kalyanmoy Deb, Lothar Thiele, Marco Laumanns, and Eckart Zitzler. Scalable multi-objective optimization test problems. In *Proceedings of the 2002 Congress on Evolutionary Computation*, volume 1, pages 825–830. IEEE, May 2002.
- [28] Tamal K. Dey. *Curve and Surface Reconstruction: Algorithms with Mathematical Analysis*. Cambridge University Press, New York, NY, USA, 2006.
- [29] Qiang Du, Vance Faber, and Max Gunzburger. Centroidal Voronoi tessellations: Applications and algorithms. *SIAM Review*, 41(4):637–676, December 1999.
- [30] David Duke, Hamish Carr, Aaron Knoll, Nicolas Schunck, Hai Ah Nam, and Andrzej Staszczak. Visualizing nuclear scission through a multifield extension of topological analysis. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2033–2040, October 2012.
- [31] Herbert Edelsbrunner, David G. Kirkpatrick, and Raimund Seidel. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, 29(4):551–559, July 1983.
- [32] Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. In *29th Conference on Learning Theory*, pages 907–940. Association for Computational Learning, June 2016.
- [33] Steven K. Feiner and Clifford Beshers. Worlds within worlds: Metaphors for exploring N-dimensional virtual worlds. In *Proceedings of the 3rd annual ACM SIGGRAPH symposium on User interface software and technology (UIST’90)*, pages 76–83. ACM, October 1990.
- [34] Thomas A. Funkhouser and Carlo H. Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments.

- In *Proceedings of SIGGRAPH 93*, Annual Conference Series, pages 247–254. ACM, September 1993.
- [35] Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, and Kiem-Phong Vo. A technique for drawing directed graphs. *IEEE Transactions on Software Engineering*, 19(3):214–230, March 1993.
 - [36] Samuel Gerber, Peer-Timo Bremer, Valerio Pascucci, and Ross Whitaker. Visual exploration of high dimensional scalar functions. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1271–1280, November/December 2010.
 - [37] Samuel Gerber and Kristin Potter. Data analysis with the Morse-Smale complex: The msr package for R. *Journal of Statistical Software*, 50(2):1–22, July 2012.
 - [38] Michael Gleicher. A framework for considering comprehensibility in modeling. *Big Data*, 4(2):75–88, June 2016.
 - [39] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
 - [40] Samuel Gratzl, Alexander Lex, Nils Gehlenborg, Hanspeter Pfister, and Marc Streit. LineUp: Visual analysis of multi-attribute rankings. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2277–2286, December 2013.
 - [41] Attila Gyulassy, Natallia Kotava, Mark Kim, Charles D. Hansen, Hans Hagen, and Valerio Pascucci. Direct feature visualization using morse-smale complexes. *IEEE Transactions on Visualization and Computer Graphics*, 18(9):1549–1562, September 2012.
 - [42] Kyle W.M. Hall, Peter G. Kusalik, and Sheelagh Carpendale. Profile contour plots: Alternate projections of 3D free energy surfaces. VisWeek 2014 Poster Compendium, 2014.

- [43] John H. Halton. Algorithm 247: Radical-inverse quasi-random point sequence. *Communications of the ACM*, 7(12):701–702, December 1964.
- [44] Andrew J. Hanson and Robert A. Cross. Interactive visualization methods for four dimensions. In *IEEE Conference on Visualization, 1993*, pages 196–203. IEEE, October 1993.
- [45] Mark A. Harrower and Cynthia A. Brewer. ColorBrewer.org: An online tool for selecting color schemes for maps. *The Cartographic Journal*, 40(1):27–37, June 2003.
- [46] John A. Hartigan. Printer graphics for clustering. *Journal of Statistical Computation and Simulation*, 4(3):187–213, June 1975.
- [47] Jeffrey Heer and Michael Bostock. Crowdsourcing graphical perception: Using mechanical turk to assess visualization design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 203–212. ACM, 2010.
- [48] Jeffrey Heer and Ben Shneiderman. Interactive dynamics for visual analysis. *Queue*, 10(2):1–26, February 2012.
- [49] Julian Heinrich and Daniel Weiskopf. Continuous parallel coordinates. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1531–1538, November/December 2009.
- [50] James Hensman, Nicolo Fusi, and Neil D. Lawrence. Gaussian processes for big data. In *Proceedings of the 29th Conference on Uncertainty in Artificial Intelligence*, pages 282–290. AUAI Press, August 2013.
- [51] Paul Hoffman, Georges Grinstein, Kenneth Marx, Ian Grosse, and Eugene Stanley. DNA visual and analytic data mining. In *Proceedings of the 8th conference on Visualization '97*, pages 437–441. IEEE Computer Society, October 1997.
- [52] Richard Holbrey. Dimension reduction algorithms for data mining and visualization. Technical report, University of Leeds/Edinburgh, 2006.

- [53] Wei Hong, Neophytos Neophytou, Klaus Mueller, and Arie Kaufman. *Mathematical Foundations of Scientific Visualization, Computer Graphics, and Massive Data Exploration*, chapter Constructing 3D elliptical Gaussians for irregular data, pages 213–225. Mathematics and Visualization. Springer Berlin Heidelberg, 2006.
- [54] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.
- [55] Peter J. Huber. Projection pursuit. *The Annals of Statistics*, 13(2):435–475, June 1985.
- [56] Lars Huettenberger, Christian Heine, and Christoph Garth. Decomposition and simplification of multivariate data using Pareto sets. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2684–2693, December 2014.
- [57] Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Kevin Murphy. Time-bounded sequential parameter optimization. In *Proceedings of the 4th International Conference on Learning and Intelligent Optimization*, pages 281–298, Venice, Italy, January 2010. Springer-Verlag Berlin.
- [58] Alfred Inselberg. The plane with parallel coordinates. *The Visual Computer*, 1(2):69–91, August 1985.
- [59] Tobias Isenberg, Petra Isenberg, Jian Chen, Michael Sedlmair, and Torsten Möller. A systematic review on the practice of evaluating visualization. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2818–2827, December 2013.
- [60] Sanjini Jayaraman and Chris North. A radial focus+context visualization for multi-dimensional functions. In *Proceedings of the Conference on Visualization '02*, pages 443–450. IEEE Computer Society, October/November 2002.

- [61] Dong Hyun Jeong, Caroline Ziemkiewicz, Brian Fisher, William Ribarsky, and Remco Chang. ipca: An interactive system for pca-based visual analytics. *Computer Graphics Forum*, 28(3):767–774, June 2009.
- [62] Eser Kandogan. Star coordinates: A multi-dimensional visualization technique with uniform treatment of dimensions. In *IEEE Information Visualization Symposium, Late Breaking Hot Topics*, pages 9–12. IEEE Computer Society, 2000.
- [63] Olga Karpenko, Wilmot Li, Niloy J. Mitra, and Maneesh Agrawala. Exploded view diagrams of mathematical surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1311–1318, November 2010.
- [64] Cari G. Kaufman, Derek Bingham, Salman Habib, Katrin Heitmann, and Joshua A. Frieman. Efficient emulators of computer experiments using compactly supported correlation functions, with an application to cosmology. *The Annals of Applied Statistics*, 5(4):2470–2492, December 2011.
- [65] Steve Kieffer, Tim Dwyer, Kim Marriott, and Michael Wybrow. HOLA: Human-like orthogonal network layout. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):349–358, January 2016.
- [66] Joseph B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, March 1964.
- [67] Hsiang-Tsung Kung, Fabrizio Luccio, and Franco P. Preparata. On finding the maxima of a set of vectors. *Journal of the ACM*, 22(4):469–476, October 1975.
- [68] Dirk J. Lehmann and Holger Theisel. Optimal sets of projections of high-dimensional data. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):609–618, January 2015.
- [69] Dirk J. Lehmann and Holger Theisel. General projective maps for multidimensional data projection. *Computer Graphics Forum*, 35(2):443–453, May 2016.

- [70] M. Lichman. UCI machine learning repository, 2013.
- [71] Crystal Linkletter, Derek Bingham, Nicholas Hengartner, David Higdon, and Kenny Q. Ye. Variable selection for Gaussian process models in computer experiments. *Technometrics*, 48(4):478–490, November 2006.
- [72] Lauro Lins, James T. Klosowski, and Carlos Scheidegger. Nanocubes for real-time exploration of spatiotemporal datasets. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2456–2465, December 2013.
- [73] Zhicheng Liu, Biye Jiang, and Jeffrey Heer. imMens : Real-time visual querying of big data. *Computer Graphics Forum*, 32(3):421–430, June 2013.
- [74] Zhicheng Liu and John T. Stasko. Mental models, visual reasoning and interaction in information visualization: A top-down perspective. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):999–1008, November 2010.
- [75] Alexander V. Lotov, Vladimir A. Bushenkov, and Georgy K. Kamenev. *Interactive Decision Maps: Approximation and Visualization of Pareto Frontier*. Springer US, 2004.
- [76] Jock Mackinlay. Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics*, 5(2):110–141, April 1986.
- [77] Martin D. McKay, Richard J. Beckman, and William J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245, May 1979.
- [78] Mahsa Mirzargar, Ross T. Whitaker, and Robert M. Kirby. Curve boxplot: Generalization of boxplot for ensembles of curves. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2654–2663, December 2014.
- [79] Daniel N. Moriasi, Jeffrey G. Arnold, Michael W. Van Liew, Ronald L. Bingner, R. Daren Harmel, and Tamie L. Veith. Model evaluation guidelines for systematic quantification of accuracy in watershed simulations. *American Society of Agricultural and Biological Engineers*, 50(3):885–900, May/June 2007.

- [80] Klaus Mueller, Torsten Möller, J. Edward Swan II, Roger Crawfis, Naeem Sha-reef, and Roni Yagel. Splatting errors and antialiasing. *IEEE Transactions on Visualization and Computer Graphics*, 4(2):178–191, April 1998.
- [81] Vito M. R. Muggeo. segmented: An R package to fit regression models with broken-line relationships. *R News*, 8(1):20–25, May 2008.
- [82] Thomas Mühlbacher and Harald Piringer. A partition-based framework for building and validating regression models. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):1962–1971, December 2013. Best Paper Award.
- [83] Tamara Munzner. A nested model for visualization design and validation. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):921–928, November 2009.
- [84] Tamara Munzner. *Visualization Analysis and Design*. AK Peters Visualization Series. CRC Press, 2014.
- [85] J. Eamonn Nash and John V. Sutcliffe. River flow forecasting through concep-tual models part I: A discussion of principles. *Journal of Hydrology*, 10(3):282–290, April 1970.
- [86] John A. Nelder and Roger Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, January 1965.
- [87] Stephan Pajer, Mark Streit, Thomas Torsney-Weir, Florian Spechtenhauser, Torsten Möller, and Harald Piringer. WeightLifter: Visual weight space ex- ploration for multi-criteria decision making. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):611–620, January 2016.
- [88] Stephen E. Palmer. *Vision Science: Photons to Phenomenology*. The MIT Press, 1999.
- [89] George M. Phillips. *Interpolation and Approximation by Polynomials*, chapter Bernstein polynomials, pages 247–290. CMS Books in Mathematics. Springer, New York, NY, 2003.

-
- [90] Harald Piringer, Wolfgang Berger, and Jurgen Krasser. HyperMoVal: Interactive visual validation of regression models for real-time simulation. *Computer Graphics Forum*, 29(3):983–992, August 2010.
- [91] Ramana Rao and Stuart K. Card. The table lens : Merging graphical and symbolic representations in an interactive focus + context visualization for tabular information. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, number April, pages 318–322. ACM New York, NY, 1994.
- [92] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning*. The MIT Press, Cambridge, MA, USA, December 2006.
- [93] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “Why should I trust you?”: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144. ACM, August 2016.
- [94] Stefan Roettger. The volume library, 2017.
- [95] Howard H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3(3):175–184, January 1960.
- [96] Andrea Saltelli, Marco Ratto, Terry Andres, Francesca Campolongo, Jessica Cariboni, Debora Gatelli, Michaela Saisana, and Stefano Tarantola. *Global sensitivity analysis: The primer*. Wiley Publishing, West Sussex, England, August 2008.
- [97] Thomas J. Santner, Brian J. Williams, and William I. Notz. *The Design and Analysis of Computer Experiments*. Springer Series in Statistics. Springer New York, New York, NY, USA, July 2003.
- [98] Michael Sedlmair, Christoph Heinzl, Stefan Bruckner, Harald Piringer, and Torsten Möller. Visual parameter space analysis: A conceptual framework. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2161–2170, December 2014.





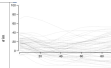
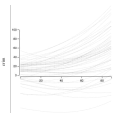
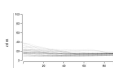
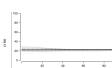
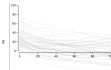
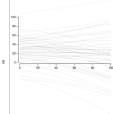
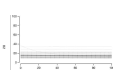
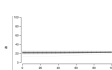

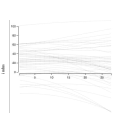
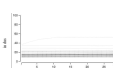
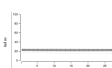

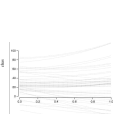


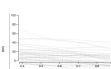
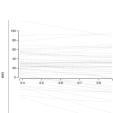


- [99] Michael Sedlmair, Miriah Meyer, and Tamara Munzner. Design study methodology: Reflections from the trenches and the stacks. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2431–2440, November/December 2012.
- [100] Tony Shepherd and Rikard Owenius. Gaussian process models of dynamic PET for functional volume definition in radiation oncology. *IEEE Transactions on Medical Imaging*, 31(8):1542–1556, August 2012.
- [101] Ben Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley Publishing Company, Reading, MA, USA, 1st edition, 1987.
- [102] Ben Shneiderman. Dynamic queries for visual information seeking. *IEEE Software*, 11(6):70–77, November 1994.
- [103] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings of the 1996 IEEE Symposium on Visual Languages*, pages 336–343, 1996.
- [104] Jeffrey S. Simonoff. *Smoothing methods in statistics*. Springer Series in Statistics. Springer-Verlag, New York, NY, USA, June 1996.
- [105] Alex J. Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222, August 2004.
- [106] John Parr Snyder. *Map projections—A working manual*. US Government Printing Office, 1987.
- [107] Il'ya Meerovich Sobol. On the distribution of points in a cube and the approximate evaluation of integrals. *USSR Computational Mathematics and Mathematical Physics*, 7(4):86–112, 1967.
- [108] Duncan Sommerville. *Introduction to the Geometry of N Dimensions*. Dover Publications, 1929.

-
- [109] Stanley S. Stevens. On the psychophysical law. *Psychological review*, 64(3):153–181, May 1957.
- [110] Boxin Tang. Orthogonal array-based latin hypercubes. *Journal of the American Statistical Association*, 88(424):1392–1397, December 1993.
- [111] Thomas Torsney-Weir, Steven Bergner, Derek Bingham, and Torsten Möller. Predicting the interactive rendering time threshold of Gaussian process models with HyperSlice. *IEEE Transactions on Visualization and Computer Graphics*, 23(2):1111–1123, February 2017.
- [112] Thomas Torsney-Weir, Ahmed Saad, Torsten Möller, Britta Weber, Hans-Christian Hege, Jean-Marc Verbavatz, and Steven Bergner. Tuner: Principled parameter finding for image segmentation algorithms using visual response surface exploration. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):1892–1901, November/December 2011.
- [113] Thomas Torsney-Weir, Michael Sedlmair, and Torsten Möller. Sliceplorer: 1D slices for multi-dimensional continuous functions. *Computer Graphics Forum*, 36(3):167–177, June 2017.
- [114] Melanie Tory, Arthur E. Kirkpatrick, M. Stella Atkins, and Torsten Möller. Visualization task performance with 2D, 3D, and combination displays. *IEEE Transactions on Visualization and Computer Graphics*, 12(1):2–13, January 2006.
- [115] Melanie Tory and Torsten Möller. Human factors in visualization research. *IEEE Transactions on Visualization and Computer Graphics*, 10(1):72–84, January 2004.
- [116] Melanie Tory and Torsten Möller. Rethinking visualization: A high-level taxonomy. In *IEEE Symposium on Information Visualization*, pages 151–158. IEEE Computer Society, October 2004.

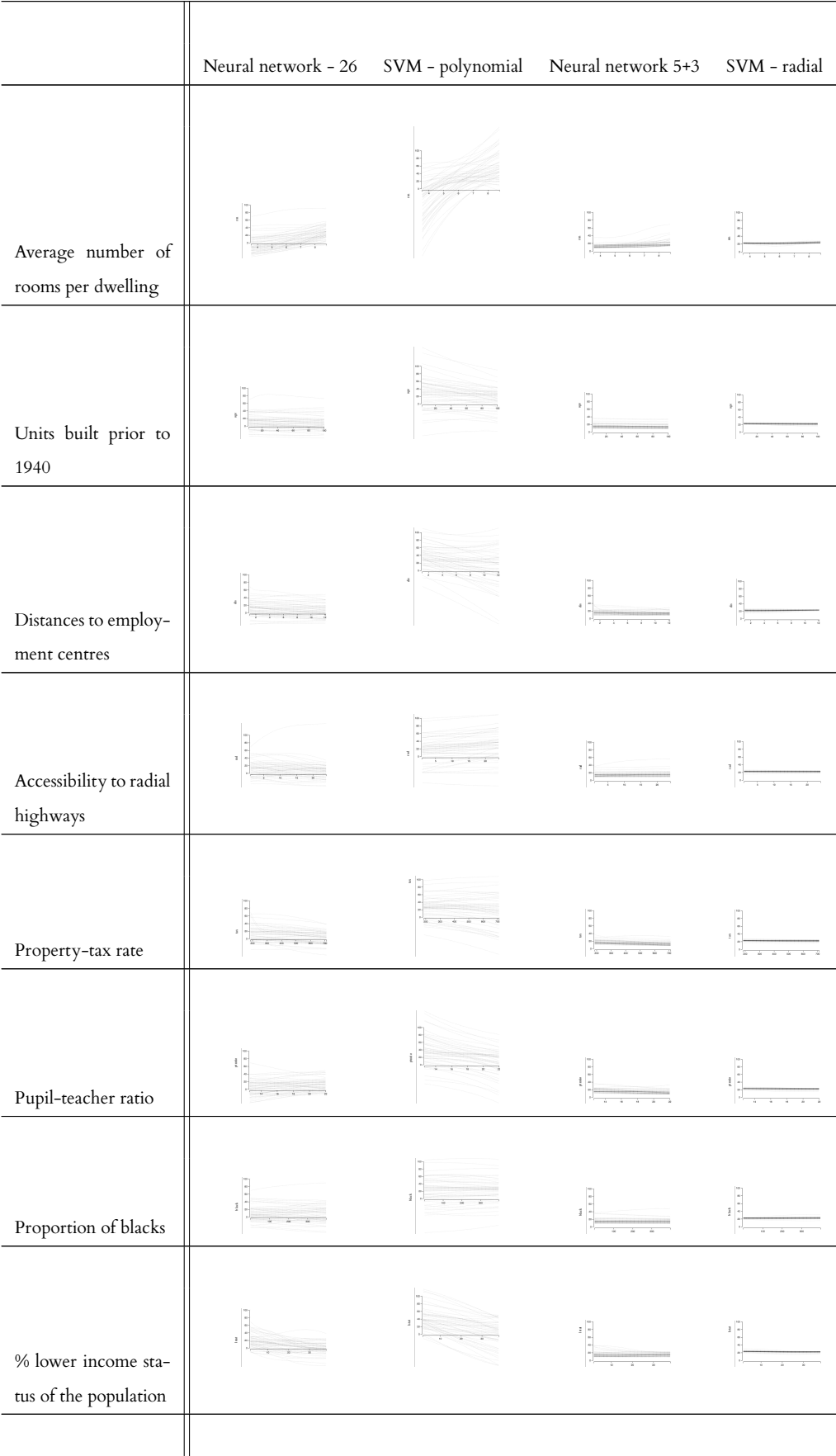
- [117] Lisa Tweedie and Robert Spence. The Prosecution Matrix: A tool to support the interactive exploration of statistical models and data. *Computational Statistics*, 13(1):65–76, March 1998.
- [118] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, November 2008.
- [119] Jarke J. van Wijk and Robert van Liere. HyperSlice: Visualization of scalar functions of many variables. In *Proceedings of the 4th Conference on Visualization*, pages 119–125. IEEE Computer Society, October 1993.
- [120] Colin Ware. *Information visualization: Perception for design*. Morgan Kaufmann, San Francisco, CA, USA, second edition, April 2004.
- [121] Robert Webb. *Stella4D 5.4*, 2017.
- [122] Rephael Wenger. *Isosurfaces: Geometry, topology, and algorithms*. A.K. Peters/CRC Press, 2013.
- [123] Ross T. Whitaker, Mahsa Mirzargar, and Robert M. Kirby. Contour boxplots: A method for characterizing uncertainty in feature sets from simulation ensembles. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2713–2722, December 2013.
- [124] Leland Wilkinson. Visualizing big data outliers through distributed aggregation. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):256–266, January 2018.
- [125] Daqing Xue and Roger Crawfis. Efficient splatting using modern graphics hardware. *Journal of Graphics Tools*, 8(3):1–21, September 2003.
- [126] Günter M. Ziegler. *Lectures on polytopes*, volume 152. Springer Science & Business Media, 2012.

A | Full sliceplorer views

Here I show the full table for Figure 2.5.

	Neural network - 26	SVM - polynomial	Neural network 5+3	SVM - radial
				
Per capita crime rate				
Residential lots over 25,000 sq.ft.				
Non-retail business acres				
1 if tract bounds river				
Nitric oxide concentration				

A. FULL SLICEPLORER VIEWS



B | The expected number of points in a parameter space

With the formulation for the expected percentage of points appearing on a slice, $\widehat{N}'(r, d)$, I now turn to the derivation of the expected number of fragments that need to be drawn per point on a slice, $\widehat{Q}(r, d)$. Only the points which pass the filtering stage of the algorithm are taken into account at this point. To do this, I derive the expected area of the quad each point will create on the slice given that the point is a certain distance, t , from the slice. Even though each point drawn leaves a circular splat, it is still represented as a quad since the GPU does not support circle primitives. To discard a sample point I generate a quad of area 0.

B.1 Expected number of fragments

Given that we need to draw a particular data point, the question is how large an impact in terms of number of fragments does it make on the 2D slice. As the distance from a sample point to the slice, t , increases the area of the quad, q , decreases. This is due to the slice passing through a smaller area of the hyperspherical kernel surrounding the data point. Figure B.1a shows the relationship between t and the half-length of one of the sides of the quad u . In Figure B.1a, (as usual) r is the maximum search distance.

Therefore, u is related to t through

$$u = \sqrt{r^2 - t^2} \tag{B.1}$$

and the maximum area of the quad is $4u^2$. However, the quad size is not always $4u^2$.

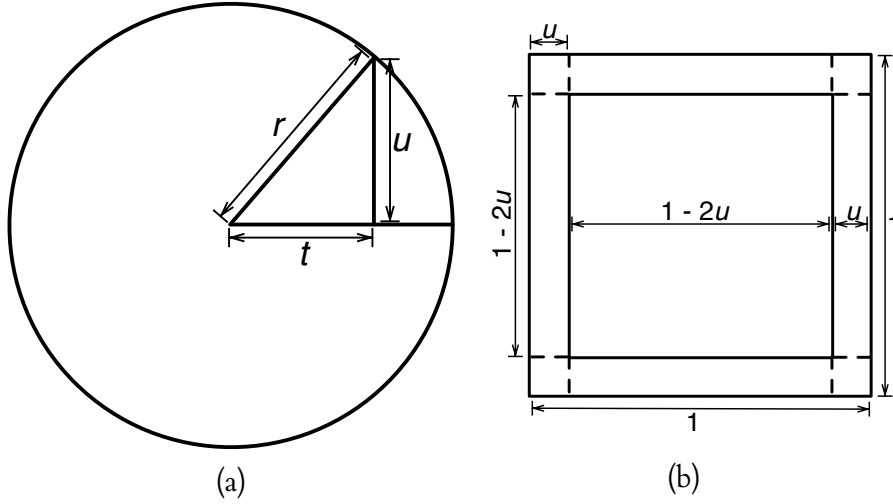


Figure B.1: (a) A 2D cross-section of a hypersphere of radius r representing the spherical kernel centered around a particular sample point. The slice we are viewing intersects the kernel at a distance, t , away. This creates an impression of side length $2u = 2\sqrt{r^2 - t^2}$ on the slice. We need to draw a quad on screen for this impression. (b) shows the possible regions on the slice (the outer square) in which the center of the sample point lies. If the sample point does not lie in the center region then some of the quad will be clipped by the edges of the screen and we will not have to render as many fragments.

If the center of the quad is within u of the edge of the slice then the quad will be clipped and it will be smaller than $4u^2$. This “maximum size” area is the inner square in Figure B.1b. We can formulate the expected quad size as a function of u : $E[q](u)$. To find $E[q](u)$ we must integrate the quad size given a location on the slice (x, y) over all possible positions of sample points,

$$E[q](u) = \int_{x=0}^1 \int_{y=0}^1 q(x, y, u) dy dx.$$

Note that there are three regions on the slice a point may fall in, the probability of the point falling into each region is a direct result of the area of each region:

- **corner:** where the quad size ranges from u^2 to $4u^2$ with probability $P = 4u^2$
- **side:** where the quad size ranges from $2u^2$ to $4u^2$ with probability $P = 4u(1 - 2u)$
- **center:** where the quad size is $4u^2$ with probability $P = (1 - 2u)^2$

Therefore, the formulation for $E[q](u)$ can be split into 3 integrals:

$$\begin{aligned}
 E[q](u) &= 4 \int_{x=0}^u \int_{y=0}^u (x+u)(y+u) dy dx \\
 &\quad + 4 \int_{x=u}^{1-u} \int_{y=0}^u (2u)(y+u) dy dx \\
 &\quad + \int_{x=u}^{1-u} \int_{y=u}^{1-u} 4u^2 dy dx \\
 &= 4A + 4B + C.
 \end{aligned}$$

Where A , B , and C are the corner, side, and center cases respectively. We solve each integral individually:

$$\begin{aligned}
 A &= \int_{x=0}^u \int_{y=0}^u (x+u)(y+u) dy dx \\
 &= \int_{x=0}^u (x+u) \int_{y=0}^u (y+u) dy dx \\
 &= \int_{x=0}^u (x+u) \left(\frac{y^2}{2} + uy \Big|_0^u \right) dx \\
 &= \int_{x=0}^u (x+u) \left(\frac{3u^2}{2} \right) dx \\
 &= \frac{3u^2}{2} \left(\frac{x^2}{2} + xu \Big|_0^u \right) \\
 A &= \frac{9u^4}{4},
 \end{aligned}$$

$$\begin{aligned}
 B &= \int_{x=u}^{1-u} \int_{y=0}^u (2u)(y+u) dy dx \\
 &= \int_{x=u}^{1-u} (2u) \int_{y=0}^u (y+u) dy dx \\
 &= 2u \int_{x=u}^{1-u} \left(\frac{y^2}{2} + yu \Big|_0^u \right) dx \\
 &= 2u \int_{x=u}^{1-u} \left(\frac{u^2}{2} + u^2 \right) dx \\
 &= 2u(1-u-u) \frac{3u^2}{2} \\
 &= 2u(1-2u) \frac{3u^2}{2} \\
 &= (2u-4u^2) \frac{3u^2}{2} \\
 B &= 3u^3 - 6u^4,
 \end{aligned}$$

and

$$\begin{aligned}
 C &= \int_{x=u}^{1-u} \int_{y=u}^{1-u} 4u^2 dy dx \\
 &= (1-u-u)(1-u-u)(4u^2) \\
 &= (1-2u)(1-2u)(4u^2) \\
 &= (1-4u+4u^2)(4u^2) \\
 C &= 4u^2 - 16u^3 + 16u^4.
 \end{aligned}$$

Substituting A , B , and C back into the formula we get,

$$\begin{aligned}
 E[q](u) &= 4A + 4B + C \\
 &= 4\frac{9u^4}{4} + 4(3u^3 - 6u^4) + (4u^2 - 16u^3 + 16u^4) \\
 &= 9u^4 + 12u^3 - 24u^4 + 4u^2 - 16u^3 + 16u^4 \\
 E[q](u) &= 4u^2 - 4u^3 + u^4. \tag{B.2}
 \end{aligned}$$

B.2 Expected 3D quad size

Now I can turn to a formulation of $\widehat{Q}(r, d)$ which is the expected quad size over all $0 \leq t \leq r$. We need to take into account the likelihood of a point lying at a particular value of t , denoted P_t :

$$\widehat{Q}(r, d) = \int_0^r E[q](u) P_t dt.$$

There are 2 cases to consider, the $d = 3$ case and the $d > 3$ case. I first derive $\widehat{Q}(r, d)$ for the simpler 3D case in order to illustrate the basic procedure combining the expected impact size with the number of points drawn.

When $d = 3$ all points lie on a line extending on either side of the slice. Therefore, $P_t = 2$, which is also the surface area of a 1-dimensional sphere. Since all distances are equally likely in this case, we can simply integrate $E[q](u)$ over all t :

$$\begin{aligned}
 \widehat{Q}(r, d) &= \int_0^r E[q](u) P_t dt \\
 &= \int_0^r (4u^2 - 4u^3 + u^4) 2 dt.
 \end{aligned}$$

We can rewrite the formula in terms of t by using Equation B.1

$$= 2 \int_0^r 4(\sqrt{r^2 - t^2})^2 - 4(\sqrt{r^2 - t^2})^3 + (\sqrt{r^2 - t^2})^4 dt.$$

To make things easier to integrate we substitute t with spherical coordinates:

$$t = r \sin \theta$$

$$dt = r \cos \theta d\theta$$

$$\sqrt{r^2 - t^2} = r \cos \theta,$$

which leads to

$$\begin{aligned} \hat{Q}(r, d) &= 2 \int_0^{\pi/2} (4(r \cos \theta)^2 - 4(r \cos \theta)^3 + (r \cos \theta)^4) r \cos \theta d\theta \\ &= 2 \int_0^{\pi/2} 4r^3 \cos^3 \theta - 4r^4 \cos^4 \theta + r^5 \cos^5 \theta d\theta \\ &= 8r^3 \int_0^{\pi/2} \cos^3 \theta d\theta - 8r^4 \int_0^{\pi/2} \cos^4 \theta d\theta + 2r^5 \int_0^{\pi/2} \cos^5 \theta d\theta \\ &= 8r^3 \left(\frac{\cos^2 \theta \sin \theta}{3} + \frac{2}{3} \int \cos \theta d\theta \right) \Big|_0^{\pi/2} \\ &\quad - 8r^4 \left(\frac{\cos^3 \theta \sin \theta}{4} + \frac{3}{4} \int \cos^2 \theta d\theta \right) \Big|_0^{\pi/2} \\ &\quad + 2r^5 \left(\frac{\cos^4 \theta \sin \theta}{5} + \frac{4}{5} \int \cos^3 \theta d\theta \right) \Big|_0^{\pi/2} \\ &= 8r^3 \left(\frac{\cos^2 \theta \sin \theta}{3} + \frac{2}{3} \sin \theta \right) \Big|_0^{\pi/2} \\ &\quad - 8r^4 \left(\frac{\cos^3 \theta \sin \theta}{4} + \frac{3}{4} \left(\frac{\theta}{2} + \frac{1}{4} \sin 2\theta \right) \right) \Big|_0^{\pi/2} \\ &\quad + 2r^5 \left(\frac{\cos^4 \theta \sin \theta}{5} + \frac{4}{5} \int \cos^3 \theta d\theta \right) \Big|_0^{\pi/2} \\ &= 8r^3 \left(\frac{2}{3} \right) - 8r^4 \left(\frac{3\pi}{16} \right) + 2r^5 \left(\frac{8}{15} \right) \\ \hat{Q}(r, d) &= \frac{16r^3}{3} - \frac{6\pi r^4}{4} + \frac{16r^5}{15}. \end{aligned}$$

B.3 Expected quad size for the $d > 3$ case

When $d > 3$, two of the dimensions are determined by the slice. However, the sample point's position with respect to the remaining $(d - 2)$ -dimensions determine

its distance to the slice. All points with the same $(d - 2)$ -dimensional distance will have the same impact on the slice. Therefore, we can view all points at distance t as lying along the surface of a hyperball. However, this hyperball may be clipped by the edges of the parameter space so the likelihood of a point lying at distance t is given by the surface area of the clipped hypersphere at t . Similar to a sphere, this can be found by taking the derivative of Equation C.11 with respect to r :

$$\begin{aligned} S(t, d) &= \frac{d\widehat{N}'(t, d)}{dt} \\ &= \frac{d \left[\sum_{i=0}^n (-1)^i \binom{n}{i} \frac{\pi^{(n-i)/2} t^{n+i}}{\Gamma(\frac{n+i}{2} + 1)} \right]}{dt} \\ &= \sum_{i=0}^n (-1)^i \binom{n}{i} \frac{(n+i) \pi^{(n-i)/2} t^{n+i-1}}{\Gamma(\frac{n+i}{2} + 1)} \\ &= \sum_{i=0}^n (-1)^i \binom{n}{i} \frac{2\pi^{(n-i)/2} t^{n+i-1}}{\Gamma(\frac{n+i}{2})}. \end{aligned}$$

We then integrate this together with the expected area of a quad (Equation B.2) over all $0 \leq t \leq r$ to find $\widehat{Q}(r, d)$:

$$\begin{aligned} \widehat{Q}(r, d) &= \int_0^r E[q](u) P_t dt \\ &= \int_0^r (4u^2 - 4u^3 + u^4) \frac{S(t, d)}{\widehat{N}'(r, d)} dt \\ &= \int_0^r \left(4(\sqrt{r^2 - t^2})^2 - 4(\sqrt{r^2 - t^2})^3 + (\sqrt{r^2 - t^2})^4 \right) \frac{S(t, d)}{\widehat{N}'(r, d)} dt \\ &= \frac{1}{\widehat{N}'(r, d)} \int_0^r \left(4(\sqrt{r^2 - t^2})^2 - 4(\sqrt{r^2 - t^2})^3 + (\sqrt{r^2 - t^2})^4 \right) S(t, d) dt \\ &= \frac{1}{\widehat{N}'(r, d)} \int_0^r \left(4(\sqrt{r^2 - t^2})^2 - 4(\sqrt{r^2 - t^2})^3 + (\sqrt{r^2 - t^2})^4 \right) \\ &\quad \left(\sum_{i=0}^n (-1)^i \binom{n}{i} \frac{2\pi^{(n-i)/2} t^{n+i-1}}{\Gamma(\frac{n+i}{2})} \right) dt. \end{aligned}$$

Just to simplify the writing a bit we can replace the factors of the formula that don't depend on t by,

$$K_i = (-1)^i \binom{n}{i} \frac{2\pi^{(n-i)/2}}{\Gamma((n+i)/2)}.$$

K_i is the expected area of the hypersphere clipped by all possible combinations of the i -dimensional space.

With this substitution we get,

$$\hat{Q}(r, d) = \frac{1}{\widehat{N}'(r, d)} \int_0^r \left(4(\sqrt{r^2 - t^2})^2 - 4(\sqrt{r^2 - t^2})^3 + (\sqrt{r^2 - t^2})^4 \right) \left(\sum_{i=0}^n K_i t^{n+i-1} \right) dt.$$

Rather than integrating over square roots, we can use trigonometric substitution to simplify the formula:

$$t = r \sin \theta$$

$$dt = r \cos \theta d\theta$$

$$\begin{aligned} \sqrt{r^2 - t^2} &= \sqrt{r^2 - (r \sin \theta)^2} \\ &= r \cos \theta. \end{aligned}$$

Then we get,

$$\begin{aligned} &= \frac{1}{\widehat{N}'(r, d)} \int_0^{\pi/2} \left(4(r \cos \theta)^2 - 4(r \cos \theta)^3 + (r \cos \theta)^4 \right) \\ &\quad \left(\sum_{i=0}^n K_i (r \sin \theta)^{n+i-1} \right) r \cos \theta d\theta \\ &= \frac{1}{\widehat{N}'(r, d)} \int_0^{\pi/2} \left(4r^3 \cos^3 \theta - 4r^4 \cos^4 \theta + r^5 \cos^5 \theta \right) \\ &\quad \left(\sum_{i=0}^n K_i r^{n+i-1} \sin^{n+i-1} \theta \right) d\theta. \end{aligned}$$

Expanding this out gives us 3 terms which are the three possible ways a sample point will show up on the slice. Each one requires some complex algebra to integrate so we handle each one as a separate derivation below.

B.3.1 Identities

During the derivation we will need the following trigonometric identities to solve our derivations in subsection B.3.2. First, some basic identities used during the computa-

tion of $I_1(m)$, $I_2(m)$, and $I_3(m)$ below,

$$\begin{aligned} \int \cos^n(\theta) \sin^m(\theta) d\theta &= \frac{\sin^{m+1}(\theta) \cos^{n-1}(\theta)}{m+n} \\ &\quad + \frac{n-1}{m+n} \int \cos^{n-2}(\theta) \sin^m(\theta) d\theta \end{aligned} \quad (\text{B.3})$$

$$\begin{aligned} \int_0^{\pi/2} \sin^m(\theta) d\theta &= -\cos(\theta) F_1\left(\frac{1}{2}, \frac{1-m}{2}, \frac{3}{2}, \cos^2(\theta)\right) \Big|_0^{\pi/2} \\ &= 0 + F_1\left(\frac{1}{2}, \frac{1-m}{2}, \frac{3}{2}, 1\right) \\ &= \frac{\sqrt{\pi} \Gamma\left(\frac{m+1}{2}\right)}{2 \Gamma\left(\frac{m}{2} + 1\right)}. \end{aligned} \quad (\text{B.4})$$

Each of the derivations in subsection B.3.2 will make use of one of the following identities based on the value of n in the $\cos^n(\theta)$ term:

$$\begin{aligned} I_1(m) &= \int_0^{\pi/2} \cos^3(\theta) \sin^m(\theta) d\theta \\ &= \frac{\sin^{m+1}(\theta) \cos^2(\theta)}{m+3} \Big|_0^{\pi/2} + \frac{2}{m+3} \int_0^{\pi/2} \cos(\theta) \sin^m(\theta) d\theta \\ &= 0 + \frac{2}{m+3} \frac{\sin^{m+1}(\theta)}{m+1} \Big|_0^{\pi/2} \\ &= \frac{2}{m+3} \left(\frac{1}{m+1} - 0 \right) \\ I_1(m) &= \frac{2}{(m+1)(m+3)}, \end{aligned} \quad (\text{B.5})$$

$$\begin{aligned}
 I_2(m) &= \int_0^{\pi/2} \cos^4(\theta) \sin^m(\theta) d\theta \\
 &= \frac{\sin^{m+1}(\theta) \cos^3(\theta)}{m+4} \Big|_0^{\pi/2} + \frac{3}{m+4} \int_0^{\pi/2} \cos^2(\theta) \sin^m(\theta) d\theta \\
 &= 0 + \frac{3}{m+4} \left[\frac{\sin^{m+1}(\theta) \cos(\theta)}{m+2} \Big|_0^{\pi/2} + \frac{1}{m+2} \int_0^{\pi/2} \sin^m(\theta) d\theta \right] \\
 &= \frac{3}{m+4} \left[0 + \frac{1}{m+2} \frac{\sqrt{\pi} \Gamma((m+1)/2)}{2\Gamma(m/2+1)} \right] \\
 &= \frac{3\sqrt{\pi}}{2(m+2)(m+4)} \frac{\Gamma((m+1)/2)}{\Gamma(m/2+1)} \\
 &= \frac{3\sqrt{\pi}}{8((m+2)/2)((m+4)/2)} \frac{\Gamma((m+1)/2)}{\Gamma(m/2+1)} \\
 &= \frac{3\sqrt{\pi}}{8(m/2+1)(m/2+2)} \frac{\Gamma((m+1)/2)}{\Gamma(m/2+1)} \\
 I_2(m) &= \frac{3\sqrt{\pi} \Gamma((m+1)/2)}{8\Gamma(m/2+3)}, \tag{B.6}
 \end{aligned}$$

and

$$\begin{aligned}
 I_3(m) &= \int_0^{\pi/2} \cos^5(\theta) \sin^m(\theta) d\theta \\
 &= \frac{\sin^{m+1}(\theta) \cos^4(\theta)}{m+5} \Big|_0^{\pi/2} + \frac{4}{m+5} \int_0^{\pi/2} \cos^3(\theta) \sin^m(\theta) d\theta \\
 &= 0 + \frac{4}{m+5} \int_0^{\pi/2} \cos^3(\theta) \sin^m(\theta) d\theta \\
 &= \frac{4}{m+5} I_1(m) \\
 &= \frac{4}{m+5} \frac{2}{(m+1)(m+3)} \\
 I_3(m) &= \frac{8}{(m+1)(m+3)(m+5)}. \tag{B.7}
 \end{aligned}$$

B.3.2 Component derivation

Derivation 1 *This integral represents the expected number of fragments drawn if the sample point is clipped by all possible combinations of the n -dimensional subspaces and the sample*

point appears in the corner of the slice we are viewing:

$$\int_0^{\pi/2} (4r^3 \cos^3 \theta) \left(\sum_{i=0}^n (-1)^i \binom{n}{i} \frac{2\pi^{(n-i)/2} r^{n+i-1} \sin^{n+i-1} \theta}{\Gamma((n+i)/2)} \right) d\theta$$

$$4 \sum_{i=0}^n (-1)^i \binom{n}{i} \frac{2\pi^{(n-i)/2} r^{n+i+2}}{\Gamma((n+i)/2)} \int_0^{\pi/2} \cos^3 \sin^{n+i-1} d\theta.$$

We can replace the integral with $I_1(n+i-1)$ from Equation B.5

$$4 \sum_{i=0}^n (-1)^i \binom{n}{i} \frac{2\pi^{(n-i)/2} r^{n+i+2}}{\Gamma((n+i)/2)} I_1(n+i-1)$$

$$4 \sum_{i=0}^n (-1)^i \binom{n}{i} \frac{2\pi^{(n-i)/2} r^{n+i+2}}{\Gamma((n+i)/2)} \frac{2}{(n+i)(n+i+2)}$$

$$4 \sum_{i=0}^n (-1)^i \binom{n}{i} \frac{2\pi^{(n-i)/2} r^{n+i+2}}{\Gamma((n+i)/2)} \frac{2}{\frac{4}{4}(n+i)(n+i+2)}$$

$$4 \sum_{i=0}^n (-1)^i \binom{n}{i} \frac{2\pi^{(n-i)/2} r^{n+i+2}}{\Gamma((n+i)/2)} \frac{2}{4(\frac{n+i}{2})(\frac{n+i}{2}+1)}$$

$$4 \sum_{i=0}^n (-1)^i \binom{n}{i} \frac{\pi^{(n-i)/2} r^{n+i+2}}{\Gamma((n+i)/2+2)}.$$

Derivation 2 This integral represents the expected number of fragments drawn if the sample point is clipped by all possible combinations of the n -dimensional subspaces and the sample point appears in the side of the slice we are viewing:

$$\int_0^{\pi/2} (4r^4 \cos^4 \theta) \left(\sum_{i=0}^n (-1)^i \binom{n}{i} \frac{2\pi^{(n-i)/2} r^{n+i-1} \sin^{n+i-1} \theta}{\Gamma((n+i)/2)} \right) d\theta$$

$$8 \sum_{i=0}^n (-1)^i \binom{n}{i} \frac{\pi^{(n-i)/2} r^{n+i+3}}{\Gamma((n+i)/2)} \int_0^{\pi/2} \cos^4 \theta \sin^{n+i-1} \theta d\theta.$$

We can replace the integral with $I_2(n+i-1)$ from Equation B.6

$$8 \sum_{i=0}^n (-1)^i \binom{n}{i} \frac{\pi^{(n-i)/2} r^{n+i+3}}{\Gamma((n+i)/2)} I_2(n+i-1)$$

$$8 \sum_{i=0}^n (-1)^i \binom{n}{i} \frac{\pi^{(n-i)/2} r^{n+i+3}}{\Gamma((n+i)/2)} \frac{3\sqrt{\pi}\Gamma((n+i)/2)}{8\Gamma((n+i-1)/2+3)}$$

$$3 \sum_{i=0}^n (-1)^i \binom{n}{i} \frac{\sqrt{\pi}\pi^{(n-i)/2} r^{n+i+3}\Gamma((n+i)/2)}{\Gamma((n+i)/2)\Gamma((n+i-1)/2+3)}$$

$$3 \sum_{i=0}^n (-1)^i \binom{n}{i} \frac{\pi^{(n-i+1)/2} r^{n+i+3}}{\Gamma((n+i+5)/2)}.$$

Derivation 3 *This integral represents the expected number of fragments drawn if the sample point is clipped by all possible combinations of the n -dimensional subspaces and the sample point appears in the center of the slice we are viewing:*

$$\int_0^{\pi/2} (r^5 \cos^5 \theta) \left(\sum_{i=0}^n (-1)^i \binom{n}{i} \frac{2\pi^{(n-i)/2} r^{n+i-1} \sin^{n+i-1} \theta}{\Gamma((n+i)/2)} \right) d\theta$$

$$2 \sum_{i=0}^n (-1)^i \binom{n}{i} \frac{\pi^{(n-i)/2} r^{n+i+4}}{\Gamma((n+i)/2)} \int_0^{\pi/2} \cos^5 \theta \sin^{n+i-1} \theta d\theta.$$

We can replace the integral with $I_3(n+i-1)$ from Equation B.7

$$2 \sum_{i=0}^n (-1)^i \binom{n}{i} \frac{\pi^{(n-i)/2} r^{n+i+4}}{\Gamma((n+i)/2)} I_3(n+i-1)$$

$$2 \sum_{i=0}^n (-1)^i \binom{n}{i} \frac{\pi^{(n-i)/2} r^{n+i+4}}{\Gamma((n+i)/2)} \frac{8}{(n+i)(n+i+2)(n+i+4)}$$

$$2 \sum_{i=0}^n (-1)^i \binom{n}{i} \frac{\pi^{(n-i)/2} r^{n+i+4}}{\Gamma((n+i)/2)} \frac{8}{8((n+i)/2)((n+i+2)/2)((n+i+4)/2)}$$

$$2 \sum_{i=0}^n (-1)^i \binom{n}{i} \frac{\pi^{(n-i)/2} r^{n+i+4}}{\Gamma((n+i)/2)} \frac{8}{8((n+i)/2)((n+i)/2+1)((n+i)/2+2)}$$

$$2 \sum_{i=0}^n (-1)^i \binom{n}{i} \frac{\pi^{(n-i)/2} r^{n+i+4}}{\Gamma((n+i)/2+3)}.$$

B.3.3 Final derivation

Putting together derivations 1–3 yields the final form for $\hat{Q}(r, d)$ for the $d > 3$ case:

$$\hat{Q}(r, d) = \frac{1}{\widehat{N}'(r, d)} \left[4 \sum_{i=0}^n (-1)^i \binom{n}{i} \frac{\pi^{(n-i)/2} r^{n+i+2}}{\Gamma((n+i)/2+2)} \right.$$

$$\quad - 3 \sum_{i=0}^n (-1)^i \binom{n}{i} \frac{3\pi^{(n-i+1)/2} r^{n+i+3}}{\Gamma((n+i+5)/2)}$$

$$\quad \left. + 2 \sum_{i=0}^n (-1)^i \binom{n}{i} \frac{\pi^{(n-i)/2} r^{n+i+4}}{\Gamma((n+i)/2+3)} \right]$$

$$= \frac{1}{\widehat{N}'(r, d)} \sum_{i=0}^n (-1)^i \binom{n}{i} \left[\frac{4\pi^{(n-i)/2} r^{n+i+2}}{\Gamma((n+i)/2+2)} - \frac{3\pi^{(n-i+1)/2} r^{n+i+3}}{\Gamma((n+i+5)/2)} + \frac{2\pi^{(n-i)/2} r^{n+i+4}}{\Gamma((n+i)/2+3)} \right].$$

Which gives the final result for $\hat{Q}(r, d)$ for a given dimension, d , and kernel radius r . This is the formula I used in the paper.

C | Derivation of $\widehat{Q}(r, d)$

For this derivation, assume a d -dimensional centred unit cube¹ $[-0.5, 0.5]^d$ with coordinate axes x_1, x_2, \dots, x_d . Without loss of generality, we specify a 2D slice by a $(d - 2)$ -dimensional (focus) point \vec{s} and assume that the two additional coordinates are the coordinates within the slice.

As explained in subsection 4.5.3, $\widehat{N}(r, d)$ is the expected percentage of points within distance r from a single 2D slice in d dimensions. In the case of a uniform point distribution, this essentially measures the volume of a slab with thickness $2r$ around the slice. The extent of this slab in the d th and $d - 1$ dimensions is one, since we have a d -dimensional unit cube. Hence, the volume of the slab is the $(d - 2)$ -dimensional volume $V(\vec{s})$ around the $(d - 2)$ -dimensional point \vec{s} multiplied by one for each direction $(d - 1)$ and d :

$$V(\vec{s}) = 1 \cdot 1 \cdot \int_{[-0.5, 0.5]^{d-2}} B_r(\vec{s} - \vec{x}) d\vec{x}, \quad (\text{C.1})$$

where, B_r is the constant $(d - 2)$ -ball with radius r :

$$B_r(\vec{x}) = \begin{cases} 1 & \text{if } \|\vec{x}\| < r \\ 0 & \text{otherwise.} \end{cases}$$

Considering the $(d - 2)$ -dimensional centred unit box $\Pi(\vec{x})$, we express Equa-

¹Note, that in chapter 4 I specified a non-centred unit cube $[0, 1]^d$. However, the centering does not impact the result, but is mathematically easier to deal with.

tion C.1 as a convolution:

$$\begin{aligned} V(\vec{s}) &= \int_{[-0.5, 0.5]^{d-2}} B_r(\vec{s} - \vec{x}) d\vec{x} \\ &= \int_{[-\infty, \infty]^{d-2}} \Pi(\vec{x}) B_r(\vec{s} - \vec{x}) d\vec{x} \\ &= \Pi * B_r(\vec{s}). \end{aligned}$$

Since $\widehat{N}'(r, d)$ is the average volume over all possible slice positions within the $(d - 2)$ -dimensional unit cube, we conclude:

$$\begin{aligned} \widehat{N}'(r, d) &= \int_{[-0.5, 0.5]^{d-2}} V(\vec{s}) d\vec{s} \\ &= \int_{[-\infty, \infty]^{d-2}} \Pi(\vec{0} - \vec{s}) \Pi * B_r(\vec{s}) d\vec{s} \\ \widehat{N}'(r, d) &= (\Pi * \Pi * B_r)(\vec{0}). \end{aligned}$$

The beauty of this last expression is, that it is a convolution of three different piecewise-constant (the constant being one) functions evaluated at zero. This allows us to reinterpret this expression as an integral of the convolution of two of these functions over the domain of the third function:

$$\widehat{N}'(r, d) = \int_{B_r} T(\vec{x}) d\vec{x}, \quad (\text{C.2})$$

where $T(\vec{x}) = \Pi * \Pi(\vec{x})$ is the convolution of two unit-cubes or the $(d - 2)$ -dimensional triangle function². Hence, in the positive orthant, we can write:

$$T^+(\vec{x}) = \prod_{i=1}^{d-2} \max(0, (1 - x_i)),$$

where $\vec{x} = (x_1, x_2, \dots, x_n)$ and, in general, $f^+(\vec{x})$ shall denote the positive orthant of some function $f(\vec{x})$.

In the remainder of this appendix we will evaluate the integral in Equation C.2 for the case of HyperSlice. Since this is an integral in a $(d - 2)$ -dimensional space, we will, for brevity, be using n to equal $(d - 2)$ in the remainder of this appendix. In the case of the HyperSlice the distance metric is the L^2 norm and the volume B_r is an

²Also known as the tensor-product of the linear B-spline or the $(d - 2)$ -linear interpolator

n -dimensional sphere of radius r . Hence, Equation C.2 becomes:

$$\widehat{N}'(r, d) = 2^n \int_{B_r^+} T^+(\vec{x}) d\vec{x}. \quad (\text{C.3})$$

We will solve this integral using polar coordinates.

C.1 Polar coordinates in n dimensions

Expressing $\vec{x} = (x_1, x_2, \dots, x_n)$ in polar coordinates $\vec{\varphi} = (R, \phi_1, \phi_2, \dots, \phi_{n-1})$ can be done as follows:

$$\begin{aligned} x_1 &= R \cos(\phi_1) \\ x_2 &= R \sin(\phi_1) \cos(\phi_2) \\ x_3 &= R \sin(\phi_1) \sin(\phi_2) \cos(\phi_3) \\ &\vdots \\ x_{n-1} &= R \sin(\phi_1) \cdots \sin(\phi_{n-2}) \cos(\phi_{n-1}) \\ x_n &= R \sin(\phi_1) \cdots \sin(\phi_{n-2}) \sin(\phi_{n-1}), \end{aligned}$$

where $\phi_i \in [0, \pi]$ for $i = 1, \dots, n-2$ and $\phi_{n-1} \in [0, 2\pi]$.

The Jacobian, needed to properly substitute the integration variables in Equation C.3 of the transformation from spatial coordinates \vec{x} to polar coordinates $\vec{\varphi}$, can be computed as follows:

$$\begin{aligned} \frac{d\vec{x}}{d\vec{\varphi}} &= R^{n-1} \prod_{i=1}^{n-1} \sin^{n-1-i}(\phi_i) \\ d\vec{x} &= R^{n-1} dR \prod_{i=1}^{n-1} \sin^{n-1-i}(\phi_i) d\phi_i. \end{aligned} \quad (\text{C.4})$$

C.2 Derivation

In this section, we will assume that the radius r never grows above one. This is a reasonable assumption, which holds for the experiments I performed. We can now

simplify Equation C.3 as follows:

$$\begin{aligned}
 \widehat{N}'(r, d) &= 2^n \int_{B_r^+} \prod_{i=1}^n \max(0, (1 - x_i)) d\vec{x} \\
 &= 2^n \int_{B_r^+} \prod_{i=1}^n (1 - x_i) d\vec{x} \\
 &= 2^n \int_{B_r^+} \sum_{i=0}^n (-1)^i t_i(\vec{x}) d\vec{x} \\
 &= 2^n \sum_{i=0}^n (-1)^i \int_{B_r^+} t_i(\vec{x}) d\vec{x}, \tag{C.5}
 \end{aligned}$$

where $t_i(\vec{x})$ is the sum of all products of i distinct coordinates. In other words,

$$\begin{aligned}
 t_0(\vec{x}) &= 1 \\
 t_1(\vec{x}) &= \sum_{j=1}^n x_j \\
 t_2(\vec{x}) &= \sum_{j,k=1; j \neq k}^n x_j x_k \\
 &\vdots
 \end{aligned}$$

Because of symmetry in the first orthant around any axis $x_i = x_j$ of the hypersphere, we have:

$$\int_{B_r^+} x_j x_k d\vec{x} = \int_{B_r^+} x_1 x_k d\vec{x} = \int_{B_r^+} x_1 x_2 d\vec{x},$$

which holds for any product of arbitrary terms. Hence, we can simplify Equation C.5 in the following way:

$$\begin{aligned}
 \widehat{N}'(r, d) &= 2^n \sum_{i=0}^n (-1)^i \int_{B_r^+} t_i(\vec{x}) d\vec{x} \\
 &= 2^n \sum_{i=1}^n (-1)^i \binom{n}{i} \int_{B_r^+} \prod_{k=1}^i x_k d\vec{x} + 2^n \int_{B_r^+} d\vec{x} \\
 &= 2^n \sum_{i=1}^n (-1)^i \binom{n}{i} A_i + B.
 \end{aligned}$$

B is the volume of an n -dimensional sphere of radius r :

$$B = 2^n \int_{B_r^+} d\vec{x} = \int_{B_r} d\vec{x} = \frac{\pi^{\frac{n}{2}} r^n}{\Gamma(\frac{n}{2} + 1)}.$$

We are left to compute the product integrals A_i , for which we use polar coordinates. For $i \leq n$, we have (using Equation C.4):

$$\begin{aligned}
A_i &= \int_{B_r^+} \prod_{k=1}^i x_k d\vec{x} \\
&= \int_{\vec{\varphi}^+} \prod_{k=1}^i x_k(\vec{\varphi}) R^{n-1} dR \prod_{j=1}^{n-1} \sin^{n-1-j}(\phi_j) d\phi_j \\
&= \int_{\vec{\varphi}^+} R^i \prod_{k=1}^i \cos(\phi_k) \sin^{i-k}(\phi_k) R^{n-1} dR \prod_{j=1}^{n-1} \sin^{n-1-j}(\phi_j) d\phi_j \\
&= \int_{\vec{\varphi}^+} R^{n+i-1} dR \prod_{k=1}^i \cos(\phi_k) \sin^{i-k}(\phi_k) \prod_{j=1}^{n-1} \sin^{n-1-j}(\phi_j) d\phi_j \\
&= \int_{\vec{\varphi}^+} R^{n+i-1} dR \prod_{k=1}^i \cos(\phi_k) \sin^{n-1+i-2k}(\phi_k) \prod_{j=i+1}^{n-1} \sin^{n-1-j}(\phi_j) d\phi_j \\
&= A_i^R \prod_{k=1}^i A_{i,k}^c \prod_{j=i+1}^{n-1} A_{i,j}^s, \tag{C.6}
\end{aligned}$$

where

$$\begin{aligned}
A_i^R &= \int_0^r R^{n+i-1} dR \\
A_{i,k}^c &= \int_0^{\pi/2} \cos(\phi_k) \sin^{n-1+i-2k}(\phi_k) \\
A_{i,j}^s &= \int_0^{\pi/2} \sin^{n-1-j}(\phi_j) d\phi_j,
\end{aligned}$$

where we used the fact that the proper positive orthant integration bounds by $\vec{\varphi}^+ = [0, R] \times [0, \pi/2]^{n-1}$.

Solving for these three types of integrals yields:

$$A_i^R = \int_0^r R^{n+i-1} dR = \frac{1}{n+i} r^{n+i},$$

$$\begin{aligned}
A_{i,k}^c &= \int_0^{\pi/2} \cos(\phi_k) \sin^{n-1+i-2k}(\phi_k) \\
&= \frac{1}{n+i-2k} \sin^{n+i-2k}(\phi_k) \Big|_0^{\pi/2} \\
&= \frac{1}{n+i-2k},
\end{aligned}$$

as well as

$$\begin{aligned}
 A_{i,j}^s &= \int_0^{\pi/2} \sin^{n-1-j}(\phi_j) d\phi_j \\
 &= -\cos(\phi_j) F_1(0.5, (j-n+2)/2, 1.5, \cos^2(\phi_j)) \Big|_0^{\pi/2} \\
 &= F_1(0.5, (j-n+2)/2, 1.5, 1) \\
 &= \frac{\Gamma(3/2)\Gamma((n-j)/2)}{\Gamma(1)\Gamma(0.5+(n-j)/2)} \\
 &= \frac{\pi^{1/2}}{2} \frac{\Gamma((n-j)/2)}{\Gamma((n-j+1)/2)},
 \end{aligned}$$

where F_1 is the hypergeometric function and Γ is the gamma function. Putting this all together (for $i \leq n$) into Equation C.6, we get:

$$\begin{aligned}
 A_i &= A_i^R \prod_{k=1}^i A_{i,k}^c \prod_{j=i+1}^{n-1} A_{i,j}^s \\
 &= \frac{1}{n+i} r^{n+i} \prod_{k=1}^i \frac{1}{n+i-2k} \prod_{j=i+1}^{n-1} \frac{\pi^{1/2}}{2} \frac{\Gamma((n-j)/2)}{\Gamma((n-j+1)/2)} \\
 &= r^{n+i} \prod_{k=0}^i \frac{1}{n+i-2k} \frac{\pi^{\frac{n-1-i}{2}}}{2^{n-1-i}} \frac{\Gamma(1/2)}{\Gamma((n-i)/2)} \\
 &= r^{n+i} \frac{\pi^{\frac{n-1-i}{2}}}{2^{n-1-i}} \frac{\pi^{1/2}}{\Gamma((n-i)/2)} \prod_{k=0}^i \frac{1}{n+i-2k} \\
 &= \frac{\pi^{\frac{n-i}{2}}}{2^{n-1-i}} \frac{r^{n+i}}{\Gamma((n-i)/2)} \prod_{k=0}^i \frac{1}{n+i-2k}. \tag{C.7}
 \end{aligned}$$

This formula can be further simplified by noting that if we expand out the product we get,

$$\begin{aligned}
 \prod_{k=0}^i \frac{1}{n+i-2k} &= \frac{1}{(n+i)(n+i-2)(n+i-4) \cdots (n+i-2i)} \\
 &= \frac{1}{\frac{2^{i+1}}{2^{i+1}} (n+i)(n+i-2)(n+i-4) \cdots (n+i-2i)} \\
 &= \frac{1}{2^{i+1} \left(\frac{n+i}{2}\right) \left(\frac{n+i}{2}-1\right) \left(\frac{n+i}{2}-2\right) \cdots \left(\frac{n+i}{2}-i\right)} \\
 &= \frac{1}{2^{i+1} \left(\frac{n+i}{2}\right) \left(\frac{n+i}{2}-1\right) \left(\frac{n+i}{2}-2\right) \cdots \left(\frac{n-i}{2}\right)}.
 \end{aligned}$$

This expansion combined with $\frac{1}{\Gamma((n-i)/2)}$ means that instead of,

$$\frac{1}{\Gamma((n-i)/2)} \prod_{k=0}^i \frac{1}{n+i-2k},$$

we can write,

$$\frac{1}{2^{i+1}\Gamma((n+i)/2)}.$$

Therefore, we can simplify Equation C.7 as,

$$\begin{aligned} A_i &= \frac{\pi^{\frac{n-i}{2}}}{2^{n-1-i}} \frac{r^{n+i}}{\Gamma((n-i)/2)} \prod_{k=0}^i \frac{1}{n+i-2k} \\ &= \frac{\pi^{\frac{n-i}{2}}}{2^{n-1-i}} \frac{r^{n+i}}{2^{i+1}\Gamma(\frac{n+i}{2}+1)} \\ &= \frac{\pi^{\frac{n-i}{2}}}{2^n} \frac{r^{n+i}}{\Gamma(\frac{n+i}{2}+1)}. \end{aligned} \quad (\text{C.8})$$

Intuitively, A_i measures the expected area of a hypersphere that is clipped by the edges of the parameter space for all possible combinations of the i -dimensional subspaces.

Finally, we can put everything together:

$$\begin{aligned} \widehat{N}'(r, d) &= 2^n \sum_{i=1}^n (-1)^i \binom{n}{i} A_i + B \\ &= 2^n \sum_{i=1}^n (-1)^i \binom{n}{i} A_i + B \\ &= 2^n \sum_{i=1}^n (-1)^i \binom{n}{i} \left[\frac{\pi^{\frac{n-i}{2}}}{2^n} \frac{r^{n+i}}{\Gamma(\frac{n+i}{2}+1)} \right] + B \\ &= \sum_{i=1}^n (-1)^i \binom{n}{i} \left[\frac{\pi^{\frac{n-i}{2}} r^{n+i}}{\Gamma(\frac{n+i}{2}+1)} \right] + \frac{\pi^{\frac{n}{2}} r^n}{\Gamma(\frac{n}{2}+1)}. \end{aligned} \quad (\text{C.9})$$

Now we note that if $i = 0$,

$$\begin{aligned} (-1)^i \binom{n}{i} \left[\frac{\pi^{\frac{n-i}{2}} r^{n+i}}{\Gamma(\frac{n+i}{2}+1)} \right] &= (-1)^0 \binom{n}{0} \left[\frac{\pi^{\frac{n-0}{2}} r^{n+0}}{\Gamma(\frac{n+0}{2}+1)} \right] \\ &= \frac{\pi^{\frac{n}{2}} r^n}{\Gamma(\frac{n}{2}+1)}. \end{aligned} \quad (\text{C.10})$$

Which is the n -dimensional volume of the ball. If we include $i = 0$ in the summation and substitute Equation C.10 into Equation C.9 then we can write $\widehat{N}'(r, d)$ as,

$$\widehat{N}'(r, d) = \sum_{i=0}^n (-1)^i \binom{n}{i} \frac{\pi^{\frac{n-i}{2}} r^{n+i}}{\Gamma(\frac{n+i}{2}+1)}. \quad (\text{C.11})$$

Abstract

Many physical data are continuous and many phenomena that we want to study are influenced by a number of factors. To understand these phenomena we need to examine multi-dimensional continuous data. Visual analysis of these phenomena can lead to many insights. However, the question remains of how to visualize something in more than three dimensions on a 2D screen. Most of the higher (i.e. more than three) dimensional data analysis tools have focused on discrete data. These methods cannot represent the full richness of the continuous process. Most continuous data visualization methods focus on either a particular domain area or a particular task (e.g. optimization).

In this thesis I explore the possibilities of creating general-purpose tools for multi-dimensional continuous data analysis. I do this through four key developments. First, I introduce a task taxonomy for continuous multi-dimensional data. Second, I investigated the use of 1D slices to understand multi-dimensional continuous functions. Third, I developed an algorithm to generate 2D slices of simplicial meshes and demonstrated how these can be used to understand shapes. Forth, I developed an algorithm to render slices in interactive time. The algorithm takes advantage of regular geometry of the multi-dimensional space as well as the GPU architecture on a modern computer.

The results of this work can be used as a basis for research on direct visualization methods of multi-dimesnsional continuous data. Through this work, I have started a discussion of the tasks involved and given concrete examples of how visualizations can be evaluated based on these tasks. My hope is that these developments will herald additional research on general methods for multi-dimensional continuous data visu-

alization.

Zusammenfassung

Kontinuierliche Daten machen einen großen Teil der physikalischen Daten aus und zahlreiche Phänomene, die man untersuchen möchte werden von unterschiedlichen Faktoren beeinflusst. Um diese Phänomene zu verstehen, müssen mehrdimensionale kontinuierliche Daten untersucht werden. Die visuelle Analyse solcher Phänomene kann viele Erkenntnisse liefern. Es stellt sich jedoch die Frage, wie man etwas in mehr als drei Dimensionen auf einem 2D-Bildschirm darstellen soll. Ein Großteil der Analysetools für hochdimensionale Daten (mehr als drei Dimensionen) konzentriert sich auf diskrete Daten. Diese Methoden können jedoch nicht die gesamte Bandbreite des kontinuierlichen Prozesses darstellen. Die meisten Methoden zur Visualisierung kontinuierlicher Daten konzentrieren sich entweder auf ein bestimmtes Gebiet oder eine bestimmte Aufgabe (z.B. Optimierung).

Im Rahmen dieser Dissertation suche ich nach Möglichkeiten, um Universaltools zur Analyse von mehrdimensionalen kontinuierlichen Daten zu schaffen. Dies tue ich mittels vier wesentlicher Problemlösungsschritte. Erstens führe ich eine Task Taxonomy für mehrdimensionale kontinuierliche Daten ein. Zweitens untersuche ich die Verwendung von 1D-Scheiben, um mehrdimensionale kontinuierliche Funktionen zu verstehen. Drittens entwickelte ich einen Algorithmus, um 2D-Scheiben von Simplicial Meshes zu erzeugen. Somit konnte ich zeigen, wie diese zum Begreifen von Formen genutzt werden können. Viertens habe ich einen Algorithmus entwickelt, um Scheiben in interaktiver Zeit zu erzeugen. Dieser Algorithmus macht sich die regelmäßige Geometrie des mehrdimensionalen Raumes zu Nutze, sowie die GPU-Architektur eines modernen Computers.

Die Ergebnisse dieser Arbeit können als Basis für Forschungen an Methoden di-

rekter Visualisierung von mehrdimensionalen kontinuierlichen Daten genutzt werden. Mit dieser Arbeit habe ich eine Diskussion über die damit verbundenen Aufgaben begonnen sowie konkrete Beispiele dafür gegeben, wie die Visualisierungen anhand dieser Aufgaben beurteilt werden können. Ich hoffe, dass die Ergebnisse dieser Arbeit zu weiteren Forschungen an allgemeinen Methoden zur Visualisierung von mehrdimensionalen kontinuierlichen Daten führen.