VISUAL ANALYSIS OF HIGH-DIMENSIONAL PARAMETER SPACES

by

Thomas Torsney-Weir

B.S., Georgetown University, 2002

A Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science

in the
School of Computing Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY
Fall 2012

APPROVAL

Name:	Thomas Torsney-Weir
Degree:	Master of Science
Title of Thesis:	Visual analysis of high-dimensional parameter spaces
Examining Committee:	Dr. Fred Popowich, Professor, Computing Science Simon Fraser University Chair
	Dr. Torsten Möller, Professor, Computing Science Simon Fraser University Senior Supervisor
	Dr. Derek Bingham, Associate Professor, Statistics and Actuarial Science Simon Fraser University Supervisor
	Dr. Greg Mori, Associate Professor, Computing Science Simon Fraser University SFU Examiner
Date Approved:	October 29, 2012

Partial Copyright Licence

Abstract

We present a system called Tuner to systematically analyze the parameter space of complex computer simulations, which are time consuming to run and consequently cannot be exhaustively sampled. We begin with a sparse initial sampling of the parameter space, then use these samples to create a fast emulator of the simulation. Analyzing this emulator gives the user insight on further sampling the simulation. Tuner guides the user through sampling and provides tools to find optimal parameter settings of up to two objective functions and perform sensitivity analysis. We present use-cases from the domain of image segmentation algorithms.

Since our method must utilize *samples* of the simulation and relies on an inherently interactive visualization method, we perform a complexity analysis to see how many samples can be rendered while staying interactive. We examined how rendering performance changes with the dimensionality, reconstruction kernel size, and number of sample points. To study this, we decomposed the rendering complexity into a predictive cost function that combines the cost of filtering each data point and then the cost to draw each pixel on screen. This cost function is calibrated to the time to filter and draw for two different hardware configurations. The cost formulation is used to examine the effects on rendering time from using box filtering versus a radial distance measure in high-dimensional data spaces as used for the filtered scatterplot and HyperSlice visualization methods, respectively. We find that for a constant kernel volume, rendering performance increases with dimensionality in the HyperSlice technique while it decreases with the filtered scatterplot technique. We also find that the total number of sample points and not the size of the reconstruction kernel is a much stronger determinant of the rendering time.

To my parents who love and support me in everything I do and encouraging me to be happy and leave my boring career to become a happy lowly student again!

"Humans are just barely intelligent tool users; Darwinian evolutionary selection stopped when language and tool use converged, leaving the average hairy meme carrier sadly deficient in smarts."

— *Accelerando*, Charles Stross, *2005*

Acknowledgements

I would like to take this opportunity to express my gratitude towards all the people who have helped and supported me in developing my degree.

First of all, I wish to thank my supervisors Dr. Torsten Möller and Dr. Derek Bingham for dedicating so much of their time to guiding me through the research process. I would have been completely lost without their constant input and advice and, of course, for pushing me to always do better. I also appreciate all the time Torsten, Derek, and my examiner Dr. Greg Mori have taken to read and help with developing my thesis.

Also, to everyone in the GrUVi lab at SFU for their advice and sitting through practice talks and half-thought-out ideas. In particular Mike, Alireza, Hamid, Usman, and Andrea. And to all my other friends who offer advice, support, and encourage me to do get away from work once in a while!

And most importantly, to my family, especially my parents, who raised me in an environment that encouraged education, exploration, and creativity. I am continually grateful for their love and support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In a diverse set of domains such as high energy particle physics, astronomy, finance, and image segmentation, experimentation via computer simulations and algorithm development is one of the driving forces in science today. Increasing developments in computer technology have opened doors previously unavailable to scientists in terms of how they conduct experiments and develop hypotheses. Before the advent of computational simulation scientists were limited by experiments they could perform in the lab. Now, simulations may provide a cheaper alternative to running experiments in the lab. For example, the CERN facility in Switzerland has a budget of 7.5bln Euros per year[1] all directed to a single experimental goal: the detection of the Higgs boson particle. Simulations permit one to describe their hypotheses as a set of machine-evaluatable formulae. These codes can then be incorporated as modules into larger systems of simulations allowing us to better algorithmically describe the natural world. In addition, one can run a number of simulation experiments in quick succession under tightly controlled conditions.

In fact, algorithms allow even "fixed" parameters such as the gravitational constant to be varied in an experiment. This has been used, for example, in the Coyote Universe simulation suite [23, 22, 31]. The goal of this simulation is to find precise predictions of the nonlinear matter power spectrum which is important in the precise detection of dark energy. The Coyote Universe is an ensemble of 38 simulated "universes" that physicists are using to match up to experimental evidence.

[1] *Large Hadron Collider — Wikipedia, the free encyclopedia*. Retrieved: August 13, 2012, from `http://en.wikipedia.org/wiki/Large_Hadron_Collider`.

Simulation modelling enables one to solve formulae that are too complex to solve analytically. Applications of the Navier-Stokes formula, for example, occur in fluid simulation, weather simulation, and aerodynamics. There is no analytic solution for Navier-Stokes which means that all "solutions" must be computed numerically through simulations. Proper analysis of these simulations in terms of how do changes in the settings of the input parameters affect the outputs is vital to successfully verifying scientific theories. In this work we present one type of system to analyze these experiments.

All of these computer codes can be characterized as a "black box" in that the simulation code is viewed as an unknown function that takes a number of inputs controlling the simulation and returns a number of outputs as a result of the simulation. These inputs and outputs may be scalar, categorical, or a complex object. For example, image segmentation algorithms take as input a source image and a number of additional parameters and return a transformed image as the result. As another example, a particle accelerator simulation produces a 3D+time output of the spatial locations of sub-atomic particles. Examining just a single instance of one of these complex output objects is not an easy task. Volume visualization and vector field visualization are active areas of research.

However, analyzing just a single experiment at a time does not allow one to infer many details about how input settings affect the output. In order to properly do this, one must run a bank of experiments, for each one varying the input parameters and recording the outputs. Now the difficulty lies in comparing a large number of outputs, evaluating their differences, and devising rules for how the parameters affect the output. This requires careful consideration of the differences between the outputs. In order to help with this, one can define a number of scalar quality measures that summarize the outputs. This allows us to examine the relationship between inputs and outputs in purely numerical means for which there are known techniques.

Using this methodology the process of understanding the behaviour of a simulation can be viewed as a kind of pipeline, shown in Fig. 1.1. We first generate *sample locations* which determine the parameter settings for a number of *simulation results*. The complex output object is then reduced via a number of *objective functions* to produce a number of scalar outputs. One then builds an emulator model of the continuous result space using an *interpolation* function, which typically involves estimating a number of modelling parameters. We can then perform *analysis* and possibly *further sample* the parameter space in order to build a better estimation model.

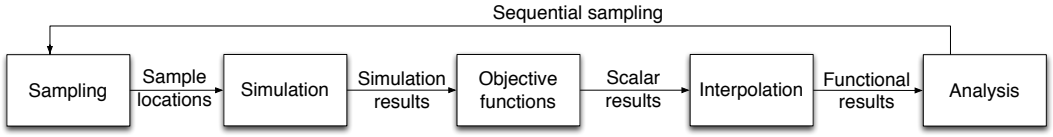


Figure 1.1: An overview of the analysis pipeline for analyzing computer simulations. In order to understand the simulation we must run a number of discrete parameter configurations, known as sampling. The results of the simulation are converted to a number of scalar measures using objective functions. We then want to interpolate these scalars to form an estimate of the *response surface*. After analyzing this continuous response surface one may take additional samples of the simulation.

In this thesis we focus primarily on the *analysis* step but also touch upon the *interpolation* and *further sampling* steps. As we will demonstrate, we feel that visualization of the parameter space best accomplishes our analysis goals.

1.1 Definitions

A number of terms will be used in this thesis and it is convenient to define them here:

Parameter space: The total range of all parameter settings controlling the simulation.

Objective function: A function that takes a complex object (e.g. an image) as input and returns a scalar output describing the "goodness" of the object.

Response surface: A manifold representing one of the (possibly many) scalar output values of the simulation.

Design sites: The locations in the parameter space for a set of discrete samples of the parameter space of the simulation. This is essentially a table containing the locations in the parameter space where we sample the simulation.

1.2 Goals

In order to decide whether it is more prudent to examine our data with, on one extreme, a purely numerical analysis or, on the other extreme, visual analysis methods, we must consider the analysis goals. Numerical methods, such as gradients or ANOVA, are advantageous when we can clearly quantify the goals of the analysis in the form of a handful of precise

measurements on data. However, there are cases where the objective measurements are not 100 percent accurate. With visual methods it is very difficult to derive precise numerical accuracy. However, one of the benefits of visualization methods is that they show the *context* of that measurement in a visual fashion in order to allow the user confirm the results of the analysis for themselves. To put it another way, when there is a clear numerical measurement that tells us everything we want to know then it's better to use that. Otherwise, a workflow using visual analysis may be best.

The goals of analysis discussed in this thesis are exploration of the full parameter space, multi-objective optimization, and analyzing the sensitivity of the objective measurements to those settings.

1.2.1 Exploration

By exploration we are referring to a user's desire to understand the full response surface. At a minimum one wants to view where in the high dimensional parameter space are the locations of "optimal" points: the local minima and maxima.

This is somewhat undirected in that the user is not interested in examining particular areas of the parameter space. Instead, they are interested in every area of the parameter space. On the surface, it seems that we would evaluate the simulation at some sufficiently small resolution to see all the behaviour we expect to see. However, these simulations are very complex and often slow in terms of time (on the order of minutes or hours) to evaluate. Therefore, it isn't feasible to run, for example, a hundred thousand samples and expect an answer in a reasonable amount of time.

There is the further cognitive difficulty of understanding a high-dimensional response surface. We live in a 3D+time-dimensional world and our perceptual limits reflect this fact. We simply do not have an intuitive grasp of what a higher-dimensional object "looks" like. We are further limited by display technology. 3D screens are available but their benefits over traditional 2D screens for data visualization purposes remains to be seen [56].

1.2.2 Multi-objective optimization

The user may also want to find the optimal parameter settings with respect to one or more objective functions. This type of analysis is called "multi-objective optimization." If we look at the computer simulation from the perspective of a functional that takes scalar inputs and

returns a number of scalar outputs (after the objective function evaluations) then the task is to find parameter settings that maximize (or minimize) this functional.

The difficulty here is that the objective measures often compete with each other. A classic example is the trade off between precision and recall measurements in classification problems [18]. Precision measures the number of positive classifications over the total number of positive results. Recall measures the number of positive classifications over the total number of possible positive classifications. Written in terms of true positive, t_p, true negative, t_n, false positive, f_p, and false negative, f_n, precision and recall are written as,

$$\text{precision} = \frac{t_p}{t_p + f_p}$$
$$\text{recall} = \frac{t_p}{t_p + f_n}$$

In addition, the weightings of these objectives may not be able to be precisely defined. In fact, the weightings may change depending on the relative value of the various objective measurements.

1.2.3 Sensitivity analysis

With any goal the user wants to know the sensitivity of the objective measures to the parameter settings. In the case of optimization, this may affect their choice of which optima to select. In the case of two closely valued optima one may choose the one that is more "stable" in the sense that changes in the parameter settings will not affect the output a great deal. This is illustrated in Fig. 1.2. What constitutes "a great deal" is application dependent.

1.3 Methods of analysis

Given these goals, the question is now, how best to analyze the input/output relationships in these computer simulations? Visualization encourages active participation of the user in performing their analysis. Furthermore, visualization methods, because the analyst is part of the feedback loop, also allow one to reason in the context of fuzzy goals such as described in Sec. 1.2. The human visual system is very good at tasks such as pattern recognition and edge detection which are still very difficult to perform algorithmically. Furthermore, looking at

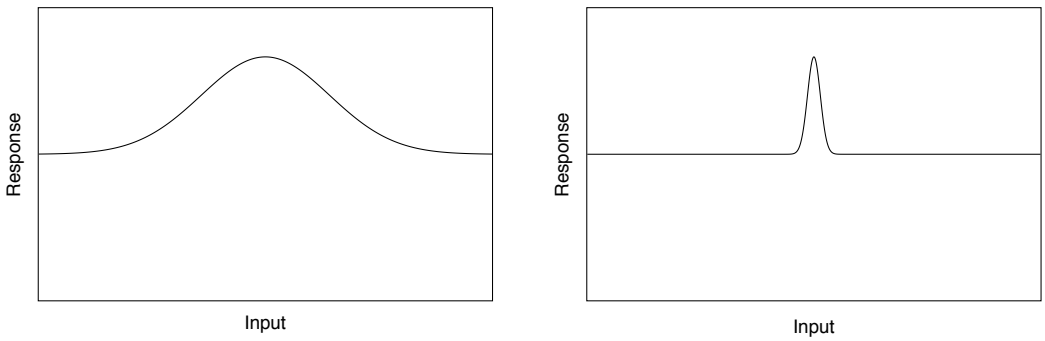


Figure 1.2: A 1D example of parameter sensitivity. The output or response value is on the y-axis and the input parameter is on the x-axis. (a) shows that the output is relatively stable with respect to changes in the parameter setting, particularly in comparison to (b) where small changes in the parameter affects the output greatly.

numerical summaries can be deceiving. A classic example is Anscombe's Quartet [1] shown in Fig. 1.3. Data in each of these four charts have the same summary statistics in terms of mean, regression formula, and variance but upon visual inspection are actually very different data sets. In this particular case one could look at the residuals from the linear regression to tell these datasets apart. However, one would have to know to do this and given that the other summary statistics are identical it is unclear if that would happen.

1.3.1 Context

During the development of Tuner (discussed in Chapter 2) we found that our users want to see not just, for example, the critical points of the response surface but also the surrounding area. Physicists and environmental scientists have also expressed interest in seeing the context of these critical points. This gives one a sense of how the response surface changes around a particular focus point. In other words, one would be able to see not just the most likely response value but also other potential responses.

1.3.2 Fuzzy goals

Visualization fits in perfectly when we have fuzzy goals by which we mean we cannot define in a precise algorithmic manner what the goal of analysis is. This does not mean this goal

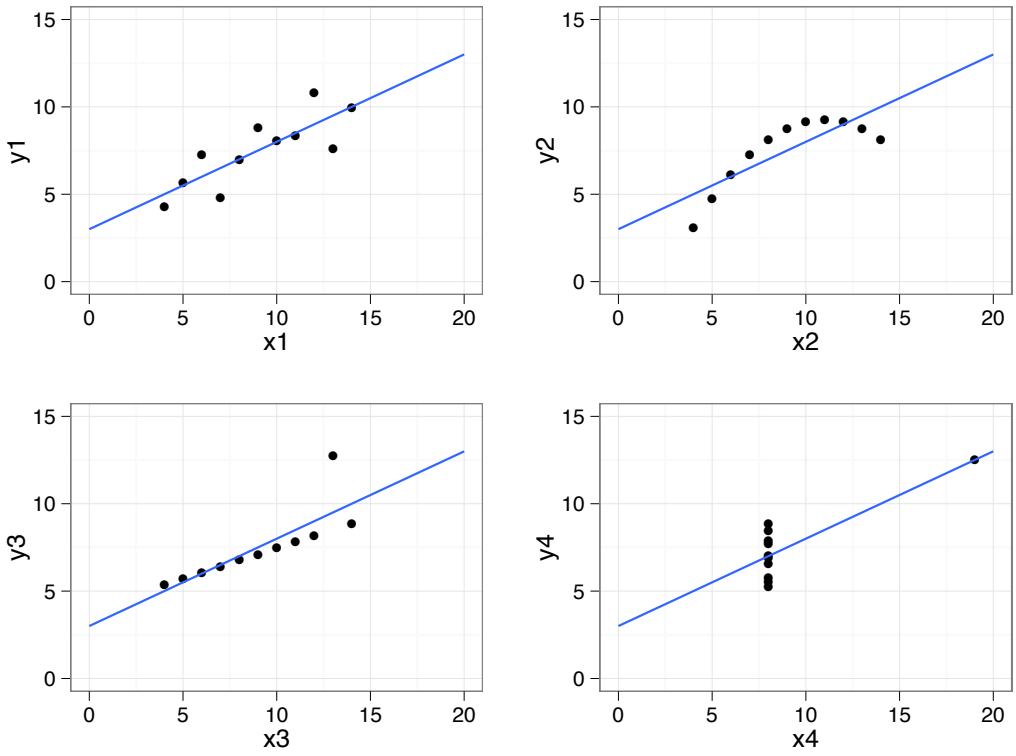


Figure 1.3: The four plots known as Anscombe's Quartet [1] demonstrating how summary statistics can be deceiving. Each of these have the same mean, regression formula, and variance despite being very different datasets.

is not achievable as the human analyst may have an exact idea of the goal, it may just be prohibitively complex to define a computable method. We believe that a human, through a proper visualization method, can very quickly evaluate a number of potential solutions and then select the best one as their analysis goal. In addition, analysis goals may change. A single visualization method may support a number of analysis goals which the human analyst can decide to use depending on the task at hand. Again, because the human is involved in making the final decision this increases confidence as they are not beholden to a fixed algorithm.

1.4 Contributions

In Chapter 2 we present Tuner, a system designed to guide the user through the entire analysis pipeline presented earlier including performing the initial sampling, analyzing the response surface, and resampling. The development of this tool focused on analysis of image segmentation algorithms. Image segmentation algorithms are characterized by being expensive to run, having complex inputs and outputs, and requiring many input parameters. Furthermore, the developers of these algorithms lacked a comprehensive solution for evaluating the sensitivity of their algorithm to the parameter settings. A key feature of Tuner is the interactive exploration of the response surface. Because maintaining this interaction is vital to understanding the relationship between parameters and outputs we then focused on how much data can Tuner handle before the interaction breaks down. In Chapter 3 we show how to calculate the expected rendering time of Tuner's HyperSlice [58] view of the response surface so that the inherently interactive exploration of the response surface can stay that way.

1.4.1 Tuner

Tuner was originally designed to address the difficult problem of parameter-finding in image segmentation. We replace a tedious manual process that is often based on guess-work and luck by a principled approach that systematically explores the parameter space. Our core idea is the following two-stage technique: We start with a sparse sampling of the parameter space and apply a statistical model to estimate each response of the segmentation algorithm. The statistical model incorporates a model of uncertainty of the estimation which we use in conjunction with the actual estimate in (visually) guiding the user towards areas that need refinement by placing additional sample points. In the second stage the user navigates through the parameter space in order to determine areas where the response values (goodness of segmentation metrics) are high. Our exploration method allows one to evaluate the tradeoffs of optimizing two response values simultaneously. We also allow the user to see the sensitivity of the response values with respect to the currently selected parameter setting. We rely on existing ground-truth images in order to evaluate a number of "goodness" measurements of an image segmentation technique. We evaluate its usefulness by demonstrating this technique on two image segmentation algorithms: a three parameter model to detect microtubules in electron tomograms and an eight parameter model to identify functional

regions in dynamic Positron Emission Tomography scans.

1.4.2 Evaluation of rendering time

We then present a complexity analysis for local visual analysis methods for multi-dimensional data sets with the display focused around a particular point of interest. Apart from enabling more confined interpretation tasks, this also provides a way to cut down rendering costs for potentially large data sets. To study this effect we decompose the rendering complexity into a predictive cost function that combines the cost of filtering each data point and then the cost to draw each pixel on screen. This cost function is calibrated to the time to filter and draw for two different hardware configurations. The cost formulation is used to examine the effects on rendering time from using box filtering versus a radial distance measure in high-dimensional data spaces as used for the Prosection matrix and HyperSlice visualization methods, respectively.

Chapter 2

Tuner

For visual analysis image data often need to be segmented. Segmentation refers to the process of partitioning the image into multiple segments, i.e. sets of pixels or voxels, that form contiguous and semantically meaningful regions. If each of these regions is marked by a unique identifier, image segmentation simply means labelling of pixels or voxels. In biomedical imaging, where images are acquired using some kind of tomography or microscopy, segmented regions might correspond to anatomical structures in the case of non-functional imaging, and to regions with specific physiological activity in the case of functional imaging.

In recent years a variety of semi- and fully automatic techniques have been developed to address the segmentation problem [41]. However, even the current state-of-the-art approaches fall short of providing a "silver bullet" for image segmentation. This has several reasons. One reason is that given some image, the segmentation problem is not well defined; in fact it depends on the application which regions are semantically meaningful. Another reason is that due to different image degradation factors such as low signal-to-noise ratio, imaging artifacts, partial volume effects and shape variability, different kinds of a priori knowledge need to be included. Additionally, the majority of the existing segmentation methods rely on and are sensitive to setting a number of parameters. For example, most of the algorithms contain weighting parameters between multiple competing image-driven or prior-driven cost terms in an attempt to mimic the cognitive capabilities of expert users (e.g. radiologists for medical images).

A good parameter setting is usually found by a manual trial and error procedure. The segmentation algorithm developer starts with a particular parameter configuration and then checks for a quality or response measure of the final segmentation measured against a ground

truth image where the correct segmentation is available. If the segmentation quality is not satisfactory, another parameter configuration will be tested. This is a tedious, time-consuming, and error-prone task. Furthermore, once a good parameter setting is found the developer then goes on (using the set of found parameters) to apply the algorithm to images without a ground truth. Because the developer has no context for the space around these ideal parameters they have no real idea of the applicability to other datasets.

Here we propose a visual analysis tool to systematically explore the multi-dimensional parameter space impacting the quality of image segmentation algorithms. We adopt a statistical model known as a Gaussian process model [44] to interpolate the response values given a sampling of the parameter space. We then use an interactive visualization to enable the exploration and refinement of the parameter space. The proposed tool can be applied to any fully automatic segmentation algorithm controlled by a number of tunable parameters and a quality measure for the obtained results.

2.1 Problem Statement

Image segmentation algorithms are typically plagued by a plethora of different tuning parameters. Conceptually, we differentiate *model parameters* from *algorithmic parameters*.

2.1.1 Model Parameters

An important class of segmentation methods are variational methods, see e.g. [38]. They rely on the minimization of an objective, or energy, functional whose minima correspond to "good" segmentations. For this class, model parameters are the weights of the different terms in the energy functional. Building an energy functional gives the algorithm designer the ability to allow multiple competing goals to be considered. The energy functional is generally formulated as follows

$$E(\phi, I) = \alpha_1 E_1(\phi, I) + \alpha_2 E_2(\phi, I) + ... + \alpha_k E_k(\phi, I), \tag{2.1}$$

where I represents the input image to be segmented, ϕ represents a segmentation, and $E_1, E_2, \ldots, E_k$ represent k different energy terms. Therefore, the parameters $\alpha_1, \alpha_2, \ldots, \alpha_k$ represent a weighting of the importance of every energy term. The final segmentation $\widehat{\phi}$ is

obtained by minimizing (Eq. 2.1) as follows:

$$\widehat{\phi} = \arg\min_{\phi} E(\phi, I). \tag{2.2}$$

For example, consider the popular Snakes algorithm [28]. Here an approximate boundary evolves to the desired boundary guided by minimizing two competing energy terms. A boundary energy term attracts the solution to pixels with high gradients. However, since boundaries optimized with only this condition tend to be jaggy due to noise in the image, an additional smoothness term is introduced to enforce the boundary of the segmented object to act like a membrane or thin plate that is trying the stretch out. Many other energy terms have been considered in the segmentation literature and we are not trying to provide a complete list here. For a good overview see, for example, Pham, Xu, and Prince [41].

2.1.2 Algorithmic Parameters

Algorithmic parameters fine-tune different parts of the algorithms. For variational segmentation, for instance, there exist approaches, like Graph Cuts [9] and Random Walker [20], where the energy term itself contains parameters to be tuned. We could describe these terms using $E(\phi, I, \sigma)$ where σ would impact how similar two nodes are that are connected through an edge.

Algorithms not based on energy minimization also have tuning parameters. For instance it is quite common to have thresholding parameters. One of the fundamental image processing algorithms is edge detection; the most popular algorithm, the Canny edge detector [12], uses three parameters, controlling the size of a Gaussian smoothing function and thresholding with hysteresis (using min/max thresholds).

In addition, almost any segmentation algorithm also includes parameters like number of iterations, accuracies for termination conditions, etc. Parameter tuning is an integral part of almost any image processing task.

2.1.3 Quality Measures

In order to produce an image segmentation, a particular parameter setting is determined and a segmentation algorithm is applied to the image. During algorithm development an expert-segmented image, or *ground truth*, is crucial to measure the quality of the segmentation. Often a visual comparison of the automatic segmentation to the expert-segmented images is

desired, but fine subtleties or 3D images are hard to inspect properly. Therefore, a number of other quality metrics have been developed.

One of the most popular metrics is the Dice similarity coefficient [16]. It measures the overlap between a segmented region and ground truth, with a value of one corresponding to a perfect overlap. Precision and Recall are two other widely used quantities to assess the quality of a classifier. In the case of image segmentation, precision measures the percentage of *true positives*, i.e., which of the segmented pixels have the right label relative to all the pixels labelled with this label by the segmentation algorithm. In contrast, Recall measures the number of correctly labelled pixels relative to all the pixels that should carry said label based on the ground truth. Ideally, both of these measures should come out to one, but often improved Precision comes at the cost of reduced Recall and vice versa. Therefore, it is useful to examine these different measures simultaneously to ensure better segmentation performance with respect to different criteria. Other commonly used tradeoffs include loss versus penalty in pattern recognition [63] and image thresholding algorithms [50]. Tuner is able to work with any segmentation technique controlled by a set of parameters and associated with a set of numerical quality measures.

2.1.4 Finding the Right Parameters

Given a segmentation model, a ground-truth, and one or several quality measures that evaluate the segmentation output relative to the ground-truth, an algorithm developer typically enters a time-consuming, tedious, and error-prone process to find good parameter values. Experience often goes a long way to come up with an initial guess. Manual variation of the parameter settings give a hint to the user of whether an improved segmentation is possible and whether the segmentation result changes slowly (i.e. we have a stable parameter region) or quickly (i.e. the segmentation result is very sensitive to the exact parameter setting). Often a single segmentation could take minutes if not tens of minutes or hours and every new parameter combination that needs to be tested will add to the frustration of the experimenter. Furthermore, keeping track of all the previously tested parameter combinations amounts to a test of patience, good memory, and being well organized. At no time of the exploration process is one ever sure, whether all the relevant parameter regions have been found. The algorithm developer has neither a systematic method for evaluating the tradeoffs among a number of objective measurements of their algorithm nor for examining the sensitivity of these objective measurements.

To facilitate the parameter exploration process, we identify a set of *tasks* that our tool needs to support. The identification of these tasks is the result of a user-centred design process. By working directly with our users during development of Tuner we were able to produce a tool that catered directly to their needs.

Exploring the full parameter space: A comprehensive and systematic way is needed to explore the full parameter space efficiently. This requires a strategy and tools for getting a quick overview and overall understanding of the parameter space in order to identify interesting regions. Furthermore, it requires means for refining the search in interesting regions.

Finding optimal parameter settings: The tool should allow the user to quickly navigate to all local optima in the global parameter space or in a sub-region of it.

Assessing the sensitivity of a parameter region: The tool should enable the user to quickly assess the sensitivity of segmentation results to parameter changes.

Simultaneous exploration of multiple quality measures: Tradeoffs between competing quality measures should be made clear and easy to explore and comprehend.

2.1.5 Contributions

Given these design constraints, we introduce a two-stage process to find optimal parameter ranges for image segmentation algorithms. During the first stage we employ an approach that samples the complete parameter space as densely as the time budget allows and then (in a batch process) automatically acquires all the corresponding segmentations. While this process is running "over-night" the user can devote his or her attention to other matters. Our approach also employs an uncertainty measure based on statistical reasoning to automatically refine regions that have not been sampled well. In the second stage the researcher explores the results of the first stage in an interactive setting. We use multidimensional navigation tools to find areas of high interest and to investigate the stability of these regions.

The contributions of Tuner can be summarized as follows: (i) We develop a systematic model to explore the full parameter space based on a Gaussian process model [27]. (ii) We allow the user to visually explore the full parameter space using sliced-based navigation (similar to HyperSlice [58]) of the response function for up-to tens of parameters. (iii) We allow the user to study the trade-off of up to two quality measures. (iv) We provide uncertainty visualization of the response surface as well as the expected gain in order to facilitate refined sampling of the parameter space.

2.2 Related work

Understanding and analyzing high-dimensional spaces has always been a challenge in statistical graphics as well as visualization. Approaches such as scatterplot matrices [14], parallel coordinates [25], and star-glyphs [59] are now common for visualizing high-dimensional data. Their main purpose, is to understand "point-clouds" or discrete entities. However, if one needs to understand *continuous* high-dimensional spaces, these approaches fail mostly, since they do not properly convey the continuity of the underlying space. Recently, Bachthaler and Weiskopf [2] extended scatterplots in order to properly portray continuous functions. However, the essence of scatterplots and similar approaches is to separate the data values from its intrinsic embedding in some metric space. This embedding is crucial if we want to understand the local sensitivity of the response surface. Sensitivity analysis studies the variance of the function to its embedding.

In medical imaging it is common to create a mental model of a 3D image of a patient by studying three orthogonal axis-aligned slices. Creating a mental model of a higher-dimensional continuous function is next to impossible, but the local behaviour of a function can be externalized leading to a cognitive relief of the user. Using a slice-plane matrix for the understanding of a high-dimensional function had been suggested by van Wijk and van Liere using a technique which they coined *HyperSlice* [58]. This idea was extended by Tweedie and Spence by what they called the Prosection Matrix [57]. Here, a thick $(n-2)$-dimensional slab is being summarized as opposed to a simple 2D slice of the n-dimensional space under study. Since we believe, that a slice will be a more accurate portrayal of the space, we use the HyperSlice approach in this paper.

While van Wijk and van Liere were inspired by the study of parameter combinations for computational steering in chemical reactions, a complete system for this study was not proposed. The idea of "seeing" into a high-dimensional parameter space in order to understand the distribution of optimal places and their sensitivity has recently found a lot of attention in the visualization community. Computational steering has been a known problem for a while, which is addressed by several researchers in the visualization community, most recently in WorldLines by Waser et al. [60]. This problem is fundamentally different from the problem we are trying to solve. In computational steering the user studies a simulation over time and actually wants to change the parameters while the simulation is running. In our case, we must set the parameters at the start of a new simulation with the intention of

optimizing the final output according to some quality measure. This is closer to the work by Bruckner and Möller in FluidExplorer [11]. However, one of the major accomplishments of FluidExplorer was dealing with simulation outputs where temporal behaviour is crucial. Furthermore, they did not address optimization of any objective function.

Approaches where an objective function is missing typically require the user to express their preference after comparing different simulation outputs (see e.g. the work by Brochu et al. [10]). This is an area also known as *active learning*. Very recently Pretorius et al. [43] have been developing a system for the exploration of parameter values for image segmentation. In their case they are not making use of any quality measures and don't assume the availability of any ground truth. Therefore, their system is quite different from ours.

Alternative approaches to facilitate parameter explorations have resulted in systems like Design Galleries [34], Image Graphs [32], spreadsheet-like exploration interfaces [26], and VisTrails [48]. None of these approaches is utilizing an optimization function nor is the user able to see a comprehensive overview of which places of the parameter space have been "looked at" and which have not.

The inspiring work by Piringer et al. [42, 5, 6] is perhaps closest to ours. Their system, *HyperMoVal*, was one of the first comprehensive environments for studying the impact of parameters on simulation experiments. HyperMoVal was also using the ideas of HyperSlice for navigating through a high-dimensional scalar function as well as facilitating a sensitivity analysis. However, HyperMoVal was geared toward industrial applications and was validated in the automobile industry. Our scope is slightly smaller and we are focused on finding good parameter combination for image segmentation. Therefore, in many ways, our problem is more constrained and requires a much less complex system. The major difference to HyperMoVal is that we use several different quality metrics in order to judge the goodness of the parameter settings. These quality measures are the basis of our exploration and simplifies the user interface immensely. In HyperMoVal, coloured contour-plots, representing the model estimation, are overlaid over scatterplots representing the measured data. Furthermore, their sensitivity analysis is very different in that it focuses on dimensional graphs by varying exactly one parameter and one local neighbourhood around a single high-dimensional parameter combination. This has been improved with the Piringer et al.'s current work [6].

2.2.1 Statistics

Our technique was born out of the work by Box and Wilson [8] in 1951. Their method is to fit gradually more and more complex estimating models to a complex function. One well-established area of research in statistics employing this idea is known as DACE - the Design and Analysis of Computer Experiments. For a good introduction we refer the reader to the book by Santner, Williams, and Notz [47]. The particular model we are using has been well described by Jones, Schonlau, and Welch [27]. It has been successfully employed in a variety of computer experiments such as an ocean circulation model [19], a hazard-effect model for volcano eruption prediction [4], and an arctic sea ice simulation [13]. However, the typical approach by statisticians is to fit the Gaussian process model and then evaluate the results of a variance decomposition. Two approaches to this method are discussed by Schonlau and Welch [49] and Oakley and O'Hagan [40]. Our approach is to provide insight by viewing and interactively exploring the full response space.

2.2.2 Automatic Parameter Tuning in Image Segmentation

A number of papers have looked into methods of automatically finding optimal parameter settings for image segmentation algorithms. Kumar and Hebert [30] applied a pseudo-likelihood technique to estimate the parameters of a conditional random field algorithm. Szummer, Kohli, and Hoiem [53] applied Graph Cuts to do maximum margin efficient learning of the segmentation parameters. Mcintosh and Hamarneh [35] optimized a non-convex energy function to find the optimal parameters. They later extended their technique using a constrained convex energy function to avoid sensitivity to the initial parameter settings [36]. In these methods, automatic parameter learning was constrained to a particular form of a segmentation technique (e.g. conditional random field or deformable model) and could not handle general algorithms as our method does. In addition, the common goal in these algorithms is to learn the parameters so that the ground truth emerges as the optimal solution. This is achieved by optimizing yet another energy function. Therefore, these techniques optimize for one particular quality measure only. In contrast, our tool provides a way to examine multiple complex responses simultaneously.

General methods for exploring parameter spaces have also been proposed. For example, Jones, Schonlau, and Welsh [27] propose an "expected gain" measurement which finds areas of the response surface likely to contain an optimum point. One can sequentially sample

where this measurement is maximal to find the most likely optimal point. Bartz-Beilstein, Lasarczyk, and Preuss [3] places the Jones, Schonlau, and Welsh work into a framework called "Sequential Parameter Optimization." Hutter et al. [24] extend this method to account for the running time of the simulation, allowing it to operate efficiently within a time budget. None of these methods, however, account for optimizing more than one objective measurement at once.

Everingham, Muller, and Thomas [17] acknowledges the issue with using a single objective measurement to measure an algorithm's performance. They used a genetic algorithm-based method for automatically finding the Pareto front of a suite of objective measurements. However, they do not evaluate how to incorporate this Pareto front into choosing optimal parameter settings. They also do not examine the sensitivity of the objective measurements. Incorporating the uncertainty from the Gaussian process model with a method like this warrants further study for automatically building up the Pareto front.

2.3 Gaussian Process Model

The core idea of our approach is to take the known segmentation results for particular parameter combinations and evaluate a number of quality metrics for these segmentations. The values from each quality metric are also known as the *response*. Knowing the responses at discrete values, we build an emulating model for each set of responses that allows us to interpolate from the known values and to estimate the quality metric at all places of the parameter space, even though we have not yet computed the actual segmentation at these places. The continuous function is often referred to as the *response surface* and the prediction at new parameter combinations is known as *inference* in statistics.

In our work, we particularly employ a *Gaussian process model* for computing the response surface. We will only be able to briefly summarize the essential ideas here and refer the reader for details to the excellent treatment by Jones, Schonlau, and Welch [27] or Santner, Williams, and Notz [47].

The Gaussian process model is a well-known technique in statistics. It assumes that the response surface is governed by some unknown random function $Y(\boldsymbol{x})$. What "random" means in this case is that we are not making any assumptions about the path of the function as long as it travels through the sample points. We are assuming that the distribution of these functions is multi-variate normal with some mean and covariance structure, however.

The covariance structure dictates the second-order behaviour of the functions. This approach allows us to estimate confidence intervals on the estimated response function [46]. The Gaussian process model interpolates a response value at an arbitrary point from known sample points (often called *design points*). One important assumption is that since we are running deterministic codes, a given input configuration will *always* produce the same output. Therefore, we don't need to model the measurement error. There is still random error from a statistical standpoint but it is in the form of estimation error from the known sample points to the continuous function. However, we don't know it ahead of time before we actually compute the segmentation and quality measure at that point. In that sense, the model of a random function $Y(\boldsymbol{x})$ encapsulates the uncertainty we have about a particular predicted response value that we have not yet computed. Hence, this uncertainty will be zero at the design points themselves.

2.3.1 Building the Model

In its most general form the model assumes that the response value at a particular parameter combination $\boldsymbol{x}$ is governed by some sort of average response function plus a deviation which is a weighted average of the response from all known sample points:

$$Y(\boldsymbol{x}) = \sum_{i=1}^{k} f_i(\boldsymbol{x})\beta_i + \epsilon(\boldsymbol{x}). \tag{2.3}$$

The first term, $\sum_{i=1}^{k} f_i(\boldsymbol{x})\beta_i$, is the regression term and the second term, $\epsilon(\boldsymbol{x})$, is the error term. The various basis functions $f_i(\boldsymbol{x})$ can be any continuous function (often low-order polynomials) of the input variable $\boldsymbol{x}$. While the choice of regression functions is often not further restricted, a common choice is to simply select the constant function only, which captures the mean behaviour, μ, of the response surface. We then allow the error term to capture any deviations off the mean. This may seem overly restrictive but it turns out that allowing the modelling to occur in the covariance structure of the error term, the restriction of the regression to the mean only does not inhibit the power of the method [47].

Assuming n design points, the error term relies on the fact, that our confidence in our estimation decreases as we move farther away from the design sites. The error at the design sites is assumed to be zero representing complete confidence in the output of a computer model. An alternative way to put this is that the error at an unsampled arbitrary location x_{new} is correlated with the design sites by some n-dimensional function $c(x_{\text{new}}, \boldsymbol{X})$ where $\boldsymbol{X}$

is an $n \times d$ matrix representing all the design sites at which we have taken samples. Again, there are a number of choices for this correlation function but a popular and effective one is the Gaussian correlation function which, for a d dimensional input, is

$$c(x^i, x^j) = \prod_{k=1}^{d} e^{\theta_k (x_k^i - x_k^j)^2}. \tag{2.4}$$

Each of the θ_k factors in the above equation are known either as *correlation parameters* or *hyperparameters*. These hyperparameters are modelled through an appropriately chosen likelihood function (which properly predicts the measured responses):

$$f(\vec{\theta}, \boldsymbol{X}) = \frac{1}{(2\pi\sigma^2)^{n/2}|\boldsymbol{R}|^{1/2}} e^{\left[-\frac{(\boldsymbol{y}-\mathbf{1}\mu)'\boldsymbol{R}^{-1}(\boldsymbol{y}-\mathbf{1}\mu)}{2\sigma^2}\right]} \tag{2.5}$$

where $\vec{\theta} = \theta_1, \theta_2, \ldots, \theta_d$. $\boldsymbol{R}$ is the $n \times n$ correlation matrix with entries $r_{i,j} = c(x^i, x^j)$ (Eq. 2.4), representing the correlation between design sites x_i and x_j taking into account the correlation parameters $\vec{\theta}$. The sample mean and variance are denoted by μ and σ^2 respectively.

The typical way to determine these hyperparameters is by maximizing this likelihood function (Eq. 2.5). This can be done numerically through an optimization procedure such as Simulated Annealing or Newton's Method. In practice this optimization converges very quickly. In our test cases this process only took a few seconds for 8 factors and up to 250 design sites.

2.3.2 Prediction

Once we have the correlation matrix $\boldsymbol{R}$ computed we can predict the response at an arbitrary point. It works out that the best linear unbiased predictor of $Y(\boldsymbol{x})$ is

$$\hat{Y}(\boldsymbol{x}_{\text{new}}) = \hat{\mu} + c(\boldsymbol{x}_{\text{new}}, \boldsymbol{X})\boldsymbol{R}^{-1}(Y - \hat{\mu}\mathbf{1}), \tag{2.6}$$

where Y is a vector of length n of the response values for each design site. We can find closed form solutions for $\hat{\mu}$ and $\hat{\sigma}^2$, the values for the mean and variance of the response surface given our correlation matrix. These are given by

$$\hat{\mu} = \frac{\mathbf{1}'\boldsymbol{R}^{-1}\boldsymbol{Y}}{\mathbf{1}'\boldsymbol{R}^{-1}\mathbf{1}} \tag{2.7}$$

and

$$\hat{\sigma}^2 = \frac{(\boldsymbol{Y} - \hat{\mu}\mathbf{1})'\boldsymbol{R}^{-1}(\boldsymbol{Y} - \hat{\mu}\mathbf{1})}{n} \tag{2.8}$$

The Gaussian process model is a statistical extension of traditional interpolation schemes, which allows the assignment of an uncertainty to the predicted value. There is no restriction to the type of basis function used, hence, spline models work just as well. It works out that the squared error in prediction, $Z^2(\boldsymbol{x})$ is

$$Z^2(\boldsymbol{x}_{\text{new}}) = \hat{\sigma}^2(\mathbf{1} - c(x_{\text{new}}, \boldsymbol{X})'\boldsymbol{R}^{-1}c(x_{\text{new}}, \boldsymbol{X})). \tag{2.9}$$

This measure reveals the confidence of the model in making a prediction at a particular point [27].

2.3.3 Next Sample Point

The squared error measure by itself is helpful if we're looking for building up an emulator for the full response surface. But, if we are simply looking for a optimum and we just sample in locations where the uncertainty is high we will sample in a number of spots that will never lead to a optimal value. Since we are interested in design points that optimize our quality measures, it would be far better to combine our current estimate at a location with the uncertainty and use that as a guide. We should sample in areas with high estimated response *and* high uncertainty.

There have been several attempts at picking the next sample point algorithmically. These methods all attempt to combine areas of high response and high uncertainty in order to find the best place at which to take more samples. We use the method outlined in Jones, Schonlau, and Welch [27] due to its ease of implementation and explanation to the user. It is very important to make sure the user can understand what the meaning is behind what the application is displaying (and it can be difficult to explain the reasoning behind a complex statistical model).

Instead of just taking the predicted point, $\hat{Y}(\boldsymbol{x}_{\text{new}})$, as a known scalar we assume that the prediction at that point is the mean of a normal distribution with standard deviation equal to the standard error of the predictor; $Z(x)$ of (Eq. 2.9). With this assumption the expected improvement at a particular point, $I(\boldsymbol{x})$, works out to be (note that E and I in

this formula are not related to (Eq. 2.1) and (Eq. 2.2))

$$E[I(\boldsymbol{x}_{\text{new}})] = (f_{\text{opt}} - \hat{Y})\Phi\left(\frac{f_{\text{opt}} - \hat{Y}}{Z(\boldsymbol{x}_{\text{new}})}\right) + Z(\boldsymbol{x}_{\text{new}})\phi\left(\frac{f_{\text{opt}} - \hat{Y}}{Z(\boldsymbol{x}_{\text{new}})}\right), \tag{2.10}$$

where $\Phi(\cdot)$ and $\phi(\cdot)$ are the cumulative normal and normal probability density functions respectively and f_{opt} is the current best response value. The first term, $(f_{\text{opt}} - \hat{Y})\Phi\left(\frac{f_{\text{opt}} - \hat{Y}}{Z(\boldsymbol{x}_{\text{new}})}\right)$, finds locations that are estimated to be better than the current optimum. Due to the correlation structure these will tend to be points close to the currently known optimum value. The second term, $Z(\boldsymbol{x}_{\text{new}})\phi\left(\frac{f_{\text{opt}} - \hat{Y}}{Z(\boldsymbol{x}_{\text{new}})}\right)$, finds areas where the level of uncertainty is high enough that we may have an optimal value despite the estimated value. These two terms tend to trade off during analysis, so a number of sample points will be selected around the currently known optimal value and then a number in areas of very high uncertainty.

2.4 Walkthrough

Here we present our system, Tuner. Tuner is designed to guide the user through the full pipeline of tuning the parameters of a segmentation algorithm. This takes them from selecting an initial sampling strategy, to finding regions of interest, to further examining these regions of interest by placing and evaluating additional samples in these regions.

The overall pipeline for the analysis is shown in Fig. 2.1. This is basically a more concrete version of the conceptual analysis pipeline presented in Fig. 1.1. The overarching idea is to start with a sparse initial sampling of the parameter space then building a more accurate model through an iterative procedure of identifying regions of interest and refining with additional samples. In this manner the user is able to identify regions that meet their criteria.

Tuner is responsible for generating the points at which to take samples. These are the *sample points* shown in Fig. 2.1. These are passed to the segmentation algorithm which assigns one or more scalar values indicating the "goodness" of segmentation. Once these *design points* are passed back to Tuner we build an interpolation model in the form of a Gaussian process model and use that model to drive the interface.

The only requirement that we impose on the segmentation code (whose inputs are being sampled) is that it can be run in the background (i.e. non-interactively). We link to the segmentation code by means of a user-specified shell script. The contract for the shell

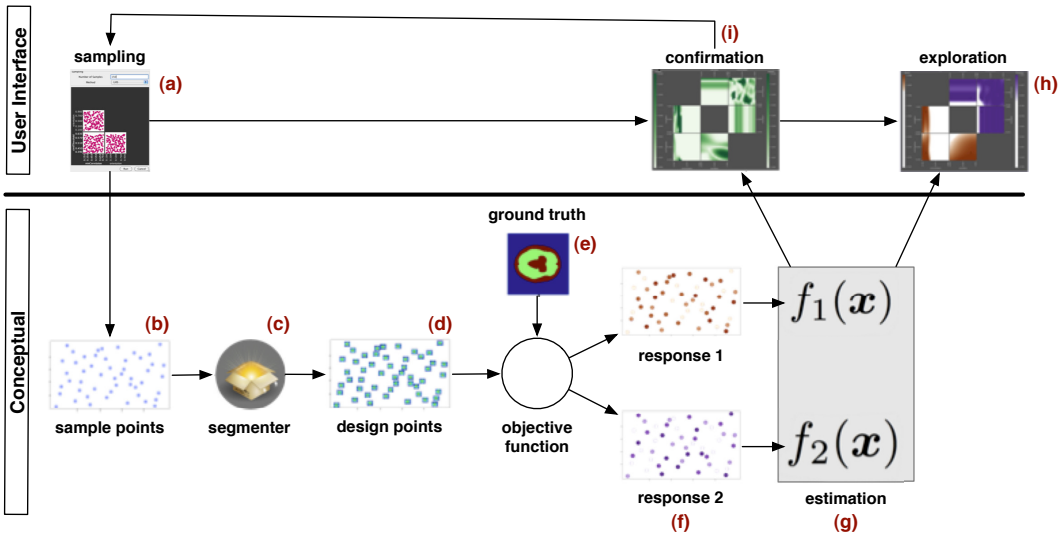
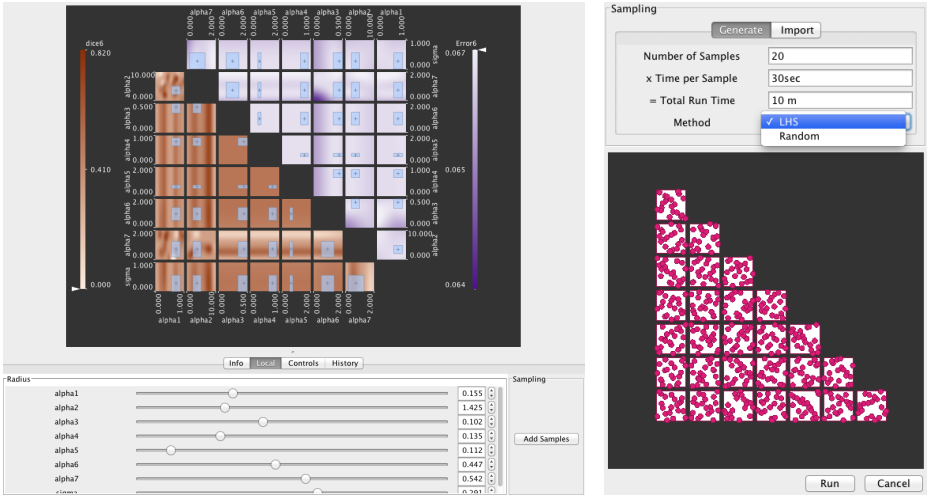


Figure 2.1: An overview of the workflow in Tuner. The user starts (a) by taking an initial sampling of the space. This generates a set of sample points (b) at which we want to compute segmentations. These points are passed off to the segmenter (c) and design points — the segmented images — are generated (d). The segmented images are compared against a ground truth image (e) in order to generate scalar responses (f). We then estimate the full response space (g) and display it to the user such that they can explore it (h). At any time the user can generate additional sample points in order to build up a more accurate model (i).

script is that it must take a reference to a file containing sample points generated by our program and write out the classified points into a file specified by Tuner. This keeps Tuner independent of any particular algorithm or platform.

2.4.1 Initial Sampling

In order to facilitate a systematic initial sampling of the parameter space we provide a workflow to place these initial samples. When the user creates a new project they are presented with the initial sampling dialog, shown in Fig. 2.2(b). Getting a dense initial sampling of the parameter space is critical for gaining a good overall estimation of the response surface, so ideally we would add as many sample points as possible. On the other hand, each sample point we add requires a run of the simulation which can take anywhere from a few minutes to a number of hours. This puts an upper-limit on the number of sample points we can evaluate in a given time. If we make no assumptions about how to weight



(a) (b)

Figure 2.2: a) Marking out an 8D hyperbox in order to place additional samples. The sliders below the box control the extent of the box centred at the current point. Adding samples brings up the sampling interface b) which uses a SPloM preview of sample point locations. The user enters the desired number of sample points and the sampling strategy and the SPloM automatically updates. The "Run" button runs the sample points directly through Tuner.

the various parameters than a uniform sampling method is often best. Perhaps the most obvious way to sample the parameter space would be to lay out a large number of samples per dimension in a Cartesian grid. However, the combination of high dimensionality of the inputs and the expense of running one sample point make this strategy prohibitively expensive if we want a dense sampling of the input space. A Cartesian grid with just 6 samples per dimension in an 8-dimensional space will yield a Cartesian grid with 6^8, or about 1.6 million, sample points. If each simulation takes 30 seconds then the total evaluation time would be around 14,000 hours. To get evaluation time down to a day would require a cluster with over 500 CPUs.

Therefore, we want to minimize the number of runs of the segmentation code while getting a good overall idea of the response manifold. As an alternative we provide the user two alternative uniform sampling strategies: Latin Hypercube sampling [37] and random sampling. Both of these strategies place an exact number of samples in the parameter space. This allows the user to accurately interpret the running time. The Latin Hypercube

in particular aims to spread sample points evenly across the range of each parameter separately [47] without the exponential increase in sample points that would occur with the Cartesian grid.

We show the generated sample points in a scatterplot matrix. This provides the user with a general idea of how well the number of sample points they have chosen fill the sampling space. Another advantage of these strategies is that both of these sampling strategies run in interactive time. When the user changes the number of sample points in the dialog they immediately see the updates in the SPloM.

Clicking on the run button begins the sampling process. Tuner monitors the state of the sampling and provides a progress bar to show feedback. Once sampling is complete Tuner automatically builds the Gaussian process model for each non-input field found.

2.4.2 Project Viewer

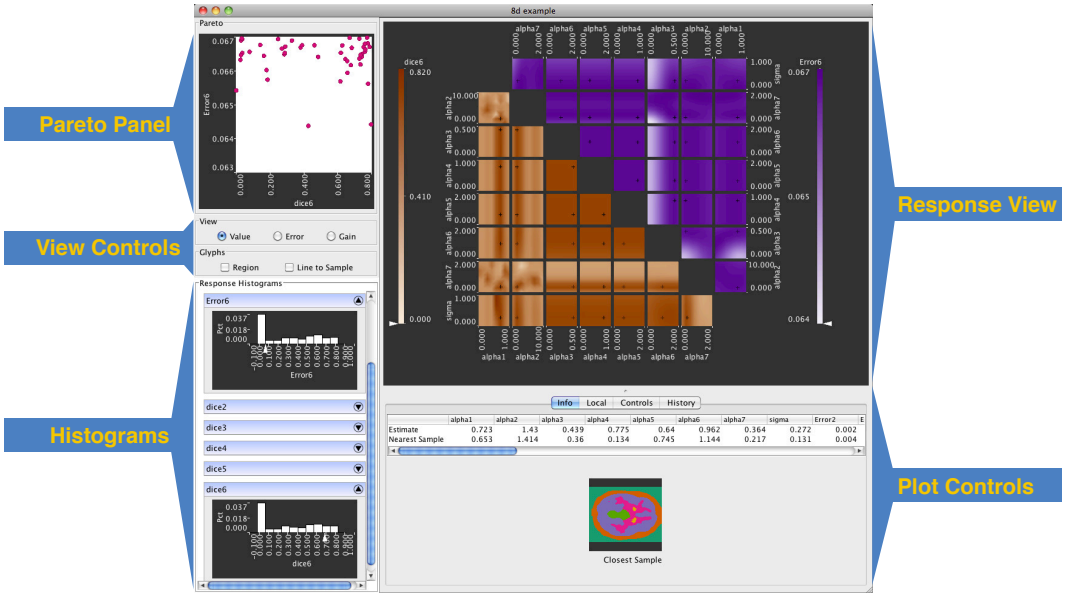


Figure 2.3: The main interaction window in Tuner. The *Response View* is a slice-based view of the response surface. The *Pareto Panel* shows the tradeoffs between the two selected response values. The *View Controls* and *Plot Controls* adjust the type of plot shown and the current parameter settings. The user can also mark out a region for sampling here. The *Histograms* show estimated histograms for each of the response variables. We have linked these views in order to facilitate the evaluation of various trade off points.

Once the Gaussian process models are built the user is presented with the Project Viewer

window, shown in Fig. 2.3. This is the primary interface for interaction with the response surface. The main sections of this interface are the *Pareto Panel*, the *Response View*, the *Controls*, and the *Histograms*. These views are designed to support the tasks of analyzing the trade off of up to two response variables, finding optimum parameter settings, and brushing and refining regions of interest.

A typical workflow with this view begins with the Pareto Panel, shown in the upper left of Fig. 2.3. The user selects a favourable output combination. They then explore the area around that known trade off point and mark out a region in which to place samples. The user then runs these sample points through the external segmentation algorithm. The Gaussian process model automatically rebuilds and the updated view is presented to the user.

2.4.2.1 Pareto Panel

Pareto analysis is an established term in statistical data analysis. It refers to the fact, that when optimizing multiple objectives, not all objectives can be optimized. The so-called *Pareto Front* are all points in parameter space where at least one objective cannot be further improved, given that all other objectives remain fixed.

The Pareto Panel, shown in Fig. 2.3 gives the user an overview of the combination of known response values. This view shows a scatterplot of the sampled response values for the selected pair of response dimensions. The optimal trade off between pairs of response values is not clearly defined. Some tasks may place a heavy weight on one particular response dimension, effectively performing a 1D optimization while other tasks may weight all response dimensions equally. Finding these optimal trade off points by navigating through the input parameter space is akin to finding a needle in a haystack.

This is an interactive tool in the sense that clicking on a particular dot in the scatterplot will set the slices to the parameters backing that dot. Thus the user can start their exploration at a known trade off point and then explore the surrounding area. We found that the users were quickly able to reason about which response to prioritize over another using this view.

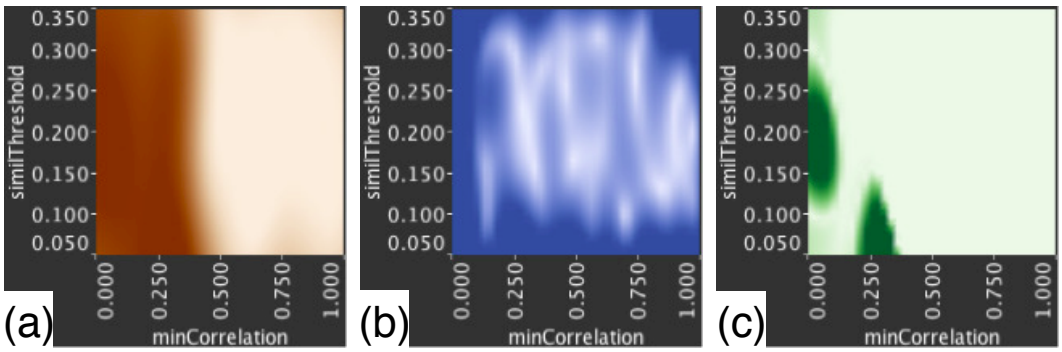


Figure 2.4: The three types of plots in our system. a) Shows the response value itself. Here darker regions indicate higher response. b) Shows the uncertainty in estimation. Here darker regions indicate higher uncertainty. c) Shows the expected gain from additional sampling. Darker colours here are areas of higher expected gain.

2.4.2.2 Response View

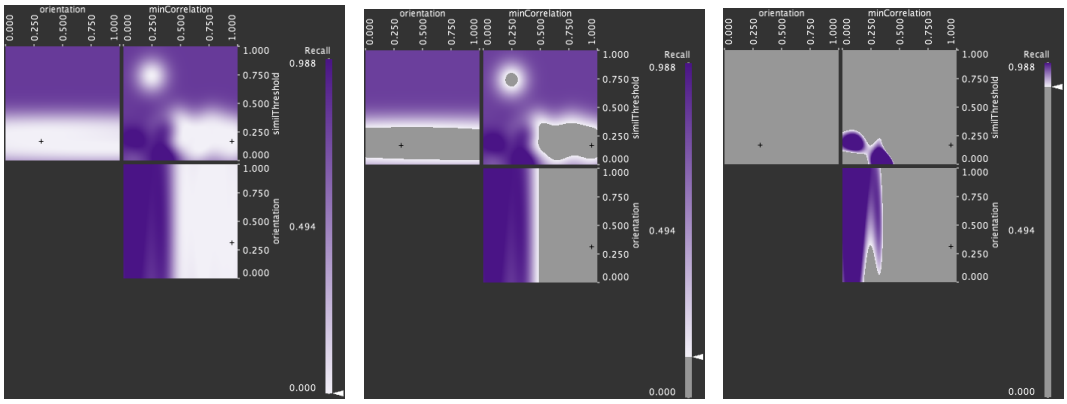
The *Response View* is the main portal for interaction with the estimated response surfaces. The plots shown are analogous to a scatterplot matrix. However, instead of scatterplots for each pair of dimensions we are showing 2D slices as was suggested in HyperSlice [58]. This is a very familiar interface for our users as slice-based views are common in medical image visualization. There are two methods for interaction with these plots. One is by clicking and dragging in the plots themselves. This changes the slices accordingly. We also provide slider controls to change the slice and range filters to change the zoom level under the "Controls" tab in the "Plot Controls" section shown in Fig. 2.3.

This view allows us to show and visually compare up to two response dimensions at a time thereby performing the multi-objective analysis task. The first response variable is shown in the lower left matrix and the second response value is shown in the upper right matrix. We use different colour maps for the different response values in order to visually distinguish them. Different response variables are distinguished through different hues using quantitative colour maps from Colorbrewer [21].

We provide three different plot types of the data to the user. An example image of each is shown in Fig. 2.4. Each of these plot types support a different task that the users want to perform. Response shows the estimated response value at any given point. This is a realization of (Eq. 2.6). This view supports finding regions of high-quality segmentation. Error shows the standard deviation (Eq. 2.9) reported by the estimation model. Here, areas

of high error are essentially gaps in our sampling. By placing sample points in these gap regions the analyst is able to build up a more accurate model across the entire input space. A model built up in this way is ideal for conducting a global sensitivity analysis. The third plot shows the expected gain in the maximum from sampling at that location. This expected gain measurement is a realization of (Eq. 2.10). By placing sample points in areas of high expected gain the user builds up a more accurate model in areas with a high likelihood of finding an optimum value. This supports the optimization task.

In practice we found that the error plots were not as useful as the gain plots. One possible explanation for this is that the error plots indicate areas of global uncertainty. This error value does not account for the fact that it may be guiding the user towards areas where the response is known to be low. However, the gain plots are taking into account the current best known sample point as well as the uncertainty. This is better suited to our optimization task since we will not spend as much time placing samples in areas where we do not expect to find any sort of optimal value.



(a) (b) (c)

Figure 2.5: An example of the interactive colorbar filter. In (a) no filter is set so the colormap is applied to all response values. In (b) a low filter value is set so only areas of the response surface below this low value are greyed out. The colormap is compressed on the remaining values. In (c) we are filtering out all but the highest values, which guides the user towards areas in the parameter space containing only the highest values.

The colour maps are interactive. The user can filter out response values that are of no interest by compressing the colour map. Fig. 2.5 shows the colormap filter at three different settings, none, low, and high levels of filtering for (a), (b), and (c) respectively. This allows

the user to better distinguish interesting from uninteresting areas. Since we are searching for optimal values, the filtering of the colormap needs to constrain the range in only one direction. Filtered values are mapped to a neutral gray colour in order to visually distinguish them from the unfiltered values.

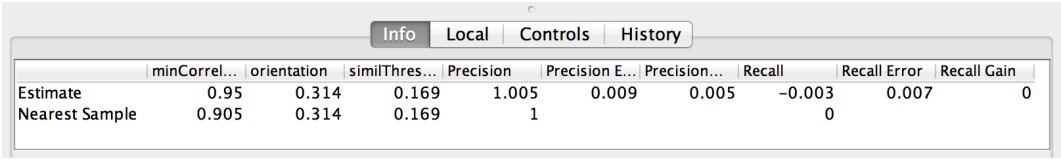
2.4.2.3 Controls

In the *Plot Controls* section (Fig. 2.6) we placed the controls that affect the user's exploration and refinement of the response surface in a tab panel. The "Info" tab shows the user the details. This includes the numerical values of all input parameters and output values for the specified focus point. It can also include an image slice of the corresponding segmentation. The focus point is also indicated with a crosshair in each plot. The "Local" tab contains a slider for each dimension. These sliders specify the size of the region of interest which can be used as a bounding box for placing additional samples or simply as a reference. This tab also contains a table listing the number of sample points in the current region as well as the gradient with respect to each input parameter. This gradient is computed as the change in response value from the current focus point to the edges of the region of interest and is displayed in the table shown at the bottom of Fig. 2.6(b). The "Controls" tab contains controls that allow the user to adjust what slice they are viewing, the zoom level, and which response variables to view. The user is also allowed to save particular points they find interesting as a type of bookmark by pressing the "H" key at any time. The "History" tab contains a table of these saved points, one per row. The user can click on a row in that table and that will take them back to that point.

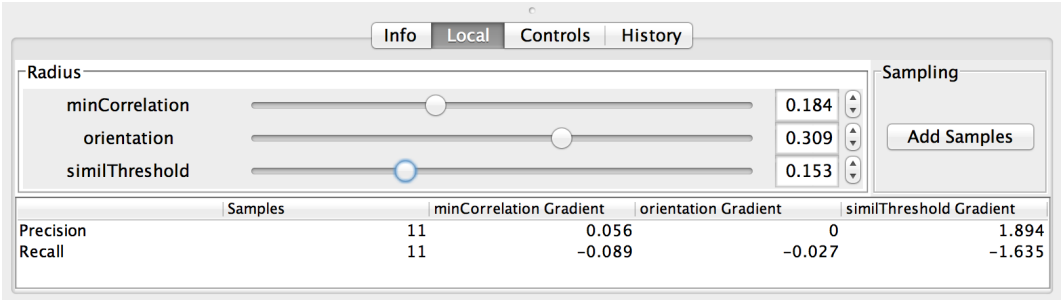
In the *View Controls* section (Fig. 2.7) we provide controls that only affect the view in the *Response View*. The plot types are changed with radio button controls. In addition, we provide controls to show and hide the currently selected region of interest and a control to show a line from the current slice to the nearest sample point. This shows users where the sample points are in the parameter space. However, neither of our users used this feature in their analysis.

2.4.2.4 Histograms

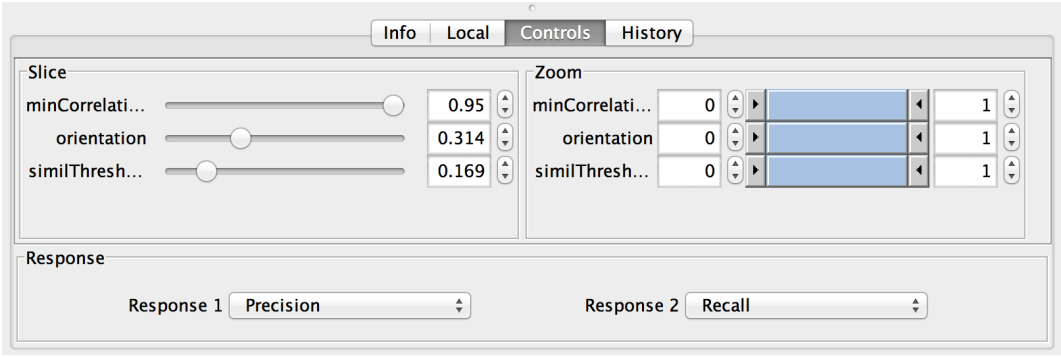
The response histograms, shown in Fig. 2.8, are designed to give the user an idea about the distribution of response values. We show one histogram per response dimension. To



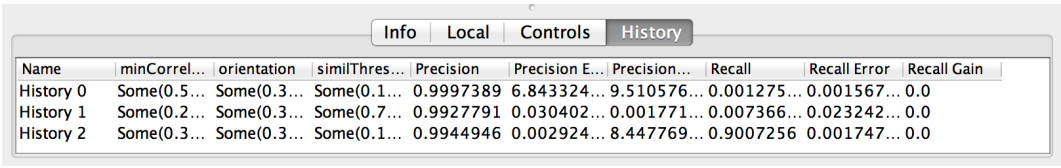
(a) Info



(b) Local

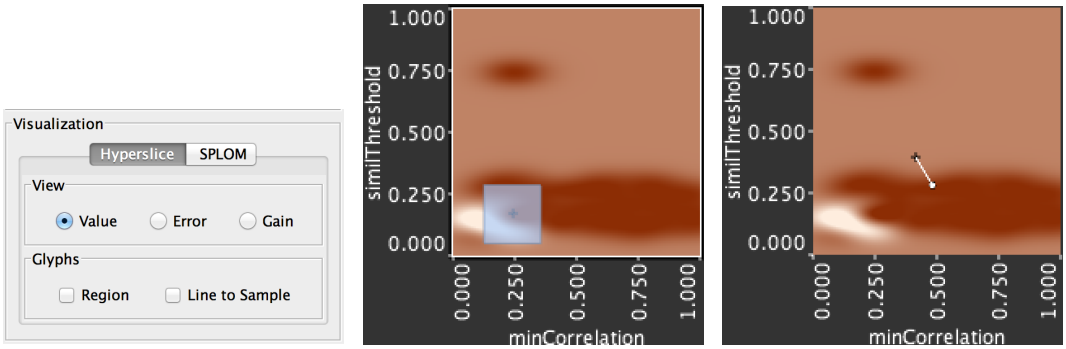


(c) Controls



(d) History

Figure 2.6: The four sections of the *Plot Controls* section of Tuner's interface. "Info," (a), displays information about the currently selected focus point. "Local," (b), displays statistics about the current region of interest. The sliders control the size of the hyperbox defining the region of interest (see Sec. 2.4.3). "Controls," (c), is an alternative interface to select the current focus point in the form of sliders as well as zoom controls which control the range of each parameter to display. "History," (d), displays a list of saved locations the user found and selected.



(a) (b) (c)

Figure 2.7: a) Controls to change what view of the response surface we are looking at. The user may also select whether to show the region of interest glyph (b) or a line drawn from the current focus point to the closest sample point (c).

generate these histograms we take a dense sampling of the *estimated* response values from the Gaussian process model.

We also use an arrow glyph in the x-axis of each histogram to show where the current slice lies in the range of outputs. This lets the user see at a glance how close the currently selected point lies in relation to all response dimensions, not only the ones shown in the Pareto Panel. The histograms can be individually hidden so that unimportant response dimensions will not clutter the interface.

2.4.3 Regions of Interest

Once the user has identified regions of interest via the response view or one of the two error views their next task is to place additional sample points in this region in so as to further refine the shape of these regions. Remember that we only took a few samples spread throughout the full high-dimensional parameter space. It is very likely that there are some regions that require a denser sampling.

The "gain" plots are well-suited to this task, an example of which is shown in Fig. 2.9(a). In this particular plot areas where the expected gain is high are represented in dark green. An advantage of using this scalar gain value for the plots is that we can utilize the same navigational interface used for finding high response to find good areas in which to take additional samples. The expected gain measure is just treated as an additional albeit always available goal function. The user does not have to learn two workflows and switching between

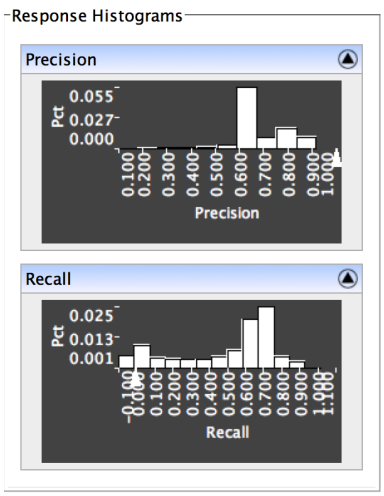


Figure 2.8: Histograms showing the distribution of response values. We show one histogram for each response variable.

sampling and response surface maximization is instantaneous.

In order to mark out a region of interest we provide the user with a set of radius sliders located under the "Local" tab in the controls section. An example of this is shown in Fig. 2.2(a). These allow the user to mark out a hyperbox centred at the current slice over the current region of interest. The user then clicks on the "Add Samples" button on that control panel which opens up a dialog that presents an interface that is identical to the initial sampling dialog, Fig. 2.2(b). The interaction here is also identical: the user selects the number of sample points to place in the region and a sampling strategy and the dialog provides a preview of the locations of the sample points in a SPloM. When the user clicks on the run button in this case the project window is closed and the user is presented with a progress bar indicating the status of the sampling. Once the sampling has finished the Project Viewer window reopens and the user may examine their refined regions.

2.4.4 Task Solutions

We conclude the walkthrough of Tuner explaining how our solutions correspond to the tasks laid out in Sec. 2.1.4.

Exploring the full parameter space: We use slice-based navigation in order to explore the full parameter space. In addition, we allow the user to click on a plot in order to snap to that location. In order to allow the user to filter out uninteresting regions we provide

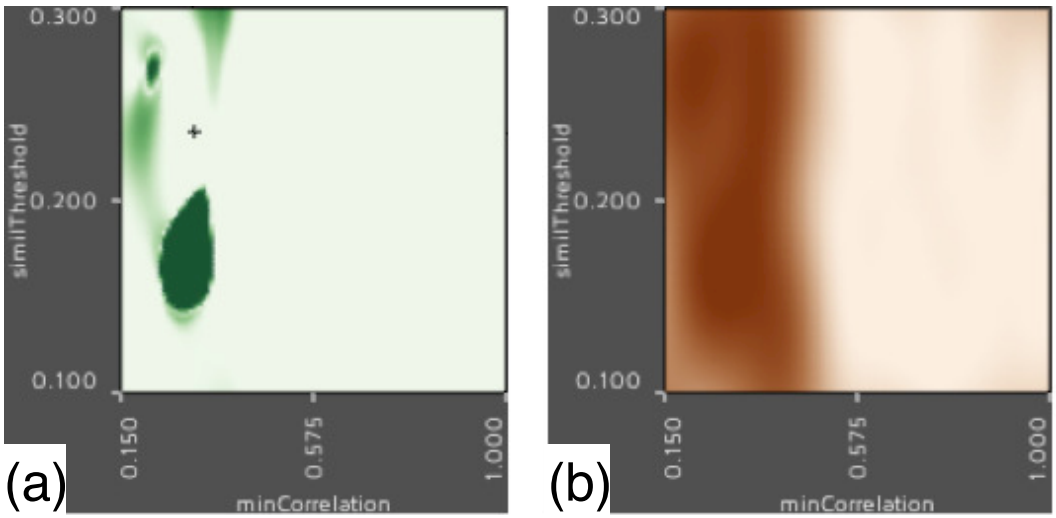


Figure 2.9: a) An example expected gain plot from our system. Note the area of high gain around minCorrelation 0.2 and similThreshold 0.175. b) The corresponding response value for the gain plot. Note that the most gain is achieved around the location of highest response.

zoom functionality in each dimension. We also allow the user to compress the colormap of either response value. Specific details about a particular point are displayed as a table to the user (details-on-demand).

Finding optimal parameter settings: Through the use of the Pareto Panel, identified in Fig. 2.3, the user is able to understand and select a trade-off point to explore in further detail. Using the expected gain plots to find areas with high expected gain around this location, the user takes additional samples in order to find the optimum value.

Assessing the sensitivity of a parameter region: As the user changes a particular parameter value all plots dependent on it change interactively. At first this may seem to be confusing for the user since so many elements are changing simultaneously. However, what the user is able to very quickly grasp is how much each plot is changing as they move through a particular dimension. The change causes their attention to automatically move to the plot that is changing the most. In fact, we found that this interactive method was preferred over looking at static slices and deciphering the effect from those.

Simultaneous exploration of multiple quality measures: We show the two response surfaces in linked views thereby showing both quality measures simultaneously. The

user is able to visualize different parameter settings and their effects on both quality measures without having to change views. We also provide a tradeoff plot for known sample points in the form of a scatterplot allowing the user to select a desired Pareto point as a starting location.

2.5 Implementation

Tuner is written in the Scala[1] programming language using APIs provided by the Processing library[2] and OpenGL. For the Gaussian process model and Latin Hypercube generator we are using packages for the R[3] environment, `mlegp` and `lhs` respectively. The link between Scala and R is provided by the JRI Java library. The axes labels were determined using the algorithm and code described in Talbot, Lin, and Hanrahan [54]. The Tuner application and source code itself is publicly available at `http://www.tomtorsneyweir.com/tuner`.

2.6 Case Studies

Our users were PhD students whose research involves the development of novel image segmentation techniques.

2.6.1 Brain Dynamic PET Study

In dynamic PET imaging, a series of 2D images are reconstructed from listmode data obtained by Gamma coincidence detectors. Kinetic modelling is the process of applying mathematical models to analyze the temporal tracer activity in order to extract clinically or experimentally relevant information. An extension of the algorithm by Saad et al. [45] is being developed by adding more energy terms to make the segmentation more robust. The goal is to segment 2D+Time PET images into six functional regions: background, skull, grey matter, white matter, cerebellum, and putamen. We used images from Saad et al. [45].

The proposed probabilistic algorithm is controlled by eight parameters: α_1 represents the weight of an image fidelity term, α_2 represents the weight of a random walker based spatial regularization term [20], α_3 represents the weight of a shape prior term, α_4 represents

[1] `http://www.scala-lang.org`

[2] `http://processing.org`

[3] `http://cran.r-project.org`

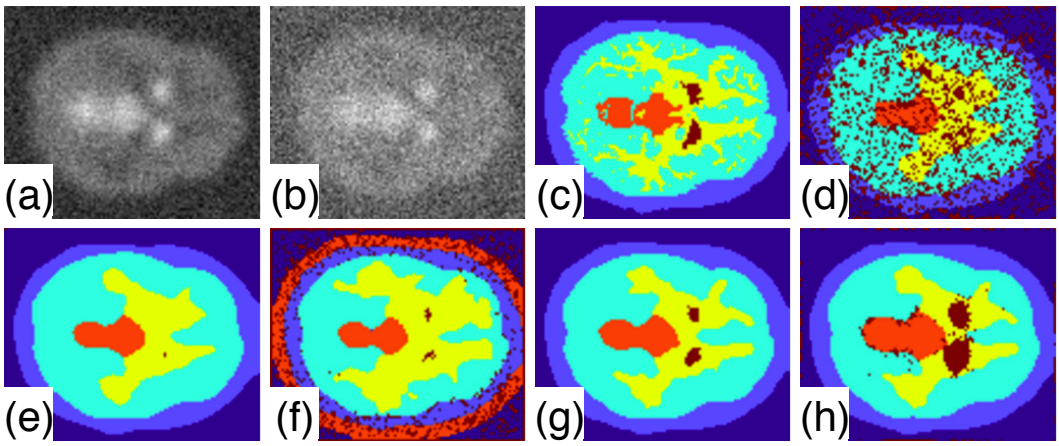


Figure 2.10: Brain dynamic PET study images. a) The last time step slice (highest signal-to-noise ratio) of the synthetic dPET data blurred with a Gaussian kernel (i.e. introduces partial volume effect) with noise level 5λ, b) with noise level 10λ, c) the ground truth segmentation, d-h) represent multiple segmentations with different parameter configurations showing the need for carefully fine-tuning the algorithm parameters.

the weight of an intensity prior term, α_5 represents the weight of a non-negativity constraint over the segmentation probability field, α_6 represents the weight of a non-negativity constraint over the recovered regional TACs, α_7 represents the weight of a prior term over the recovered regional time activity curves (TACs) using a set of templated TACs, σ represents a parameter that impacts how similar two nodes are that are connected through edges which is common in graph-based approaches [20]. Two main quality measures have been calculated for the putamen structure, the dark brown object in Fig. 2.10(c): the Dice metric [16] that measures the quality of the recovered shape and the glucose metabolic rate recovery error that measures the error in recovering the physiological parameter under investigation.

Fig. 2.10(a) and Fig. 2.10(b) show the last dPET time step with noise levels 5λ and 10λ respectively. Here, λ is used to scale the unit variance of the random noise generator to the scale of the synthetic TAC intensity at each time step [45]. We show the last time step as it has the highest signal-to-noise ratio (SNR), as is typical in dPET (the preceding time frames are even noisier). Fig. 2.10(c) shows the ground truth image that we hope to obtain. Fig. 2.10(d)-Fig. 2.10(h) show multiple segmentations with different parameter configurations demonstrating the need for fine-tuning the algorithm parameters to obtain suitable results.

Fig. 2.11 shows the exploration stages for the dPET parameter space using Tuner. We

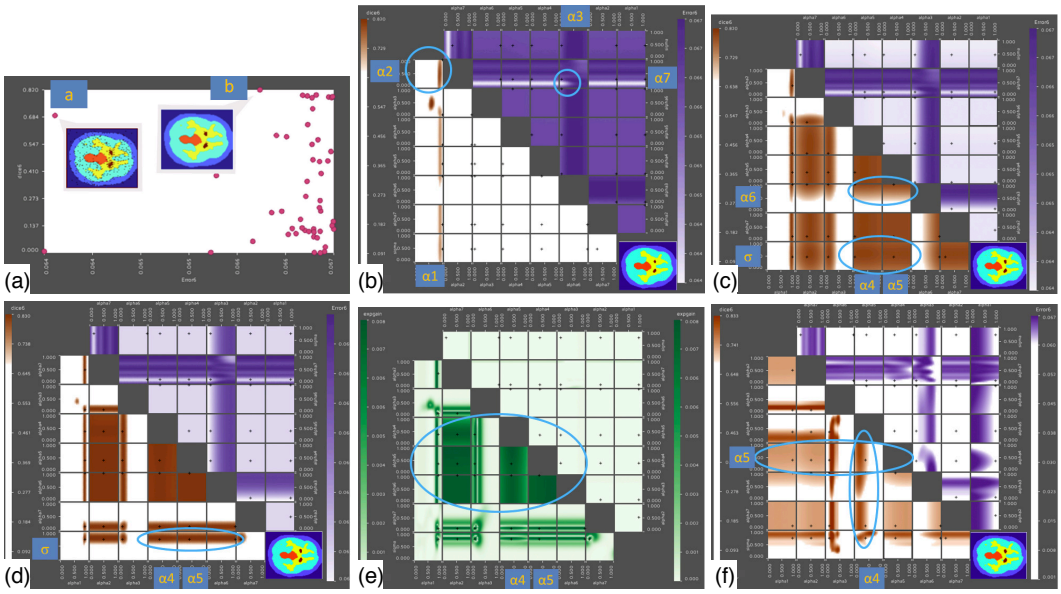


Figure 2.11: Parameter exploration of the dPET parameter space. a) Dice metric versus the glucose metabolic recovery error Pareto Panel that corresponds to 50 samples. b) slice plot matrix that corresponds to the initial parameter configuration of point b. The bottom-right corner shows the corresponding segmented image. c) Note that the response value is not very sensitive to σ, α_4, and α_5 parameters but α_6 must be set above 0.6 in order to get a good segmentation. d) shows the slice plot matrix of higher noise level 10λ e) expected gain plot suggesting more needed samples especially in the regions of α_4 and α_5. f) 250 samples slice plot matrix.

first ran an experiment with 50 different parameter configurations sampled using a Latin Hypercube method. The first goal is to obtain a parameter configuration with low glucose metabolic recovery error and high Dice metric for the putamen structure. In order to choose a starting point for our exploration, we examine the Pareto Panel showing the Dice metric versus the glucose metabolic recovery error Fig. 2.11(a). Examining the preview image at point a, shown in Fig. 2.11(a), represents a low error value with moderate Dice value but the corresponding image shows a salt-and-pepper like noise in the segmented image. Point b represents moderate error with high Dice value and a better segmented image. Hence, we choose the parameter configuration corresponding to point b as a starting configuration. Fig. 2.11(b) shows the slice plot matrix corresponding to the initial parameter configuration of point b. It shows the possibility of obtaining lower recovery error in the α_7 vs. α_3 plot and better Dice in the α_2 vs. α_1 plot. The bottom-right corner shows the respective segmented

image. Fig. 2.11(c) shows the selection of the optimal α_1, α_2, α_3, and α_7. It also shows that $\alpha_6 \geq 0.7$ is a stable region in terms of the Dice coefficient. For α_4, α_5, and σ, the Dice plot shows larger stable regions.

To examine the performance with higher noise levels, Fig. 2.11(d) examines the slice plot with additive noise of variance 10λ. It shows that there is shrinkage in the stable region in the Dice plot for σ suggesting a good range to be $0.6 \leq \sigma \leq 0.8$ but still not a definitive answer yet for α_4 and α_5. Fig. 2.11(e) shows the expected gain plot, which suggests adding more samples for this higher noise level especially in the α_4 and α_5 ranges. Fig. 2.11(f) shows a 250 sample experiment for the 10λ noise level. It shows that lower values of α_4 give higher Dice values. However, it still shows wide stable region for α_5 suggesting that neither the Dice value nor the error metric are sensitive to the change of α_5. This has to be related to the fact that other spatial regularization parameters prevent the segmentation probability matrix to contain negative values which α_5 controls directly. The final parameter configuration yields a Dice value of 0.806 and error value of 0.064.

We have applied the same values to 11 datasets accounting for inter-subject anatomical variability. We find a mean Dice value of 0.754 with standard deviation of 0.084 and a mean error value of 0.0641 with standard deviation of 0.000078. This shows that the parameter settings picked from wider stable regions in the parameter space lead to reasonable segmentations for images from the same population [35].

2.6.2 Microtubule Extraction from Electron Microscopy Tomograms

We used Tuner to determine the best parameter settings for a biological segmentation algorithm designed to extract microtubule centerlines from Electron Microscopy Tomograms. The aim of this segmentation is to give reasonable measures for the density of microtubules in the specimen. Microtubules constitute a part of the cell cytoskeleton and are subject to extensive study in biological research due to their important role in scaffolding and cell division.

The segmentation algorithm proceeds by first enhancing the centerlines of the microtubules in the tomograms by computing normalized cross-correlation with an idealized cylinder [33, 61]. In the second step the geometry of the centerlines is reconstructed using a simple iterative greedy traversal in the enhanced volume. The tracing algorithm has three parameters: a threshold, *minCorrelation*, determining where to search for microtubules in the volume (MC), a model parameter, *orientation*, penalizing strong curvature of the

traced lines (OW), and a second threshold, *similThreshold*, that determines when to stop the tracing of one line (ST). The outputs of this algorithm are polygonal lines representing microtubules in the volume.

Finding a good parameter set by visual inspection is nearly impossible since the microtubule network can be very dense, containing between 500 and 2000 lines. Many datasets ($30 - 100$) have to be processed as well. Thus we aim to find common parameter settings yielding good results on *few* test data sets and apply the found parameters to the remaining data. The assumption is that a parameter set exists that yields reasonable results for all data. Hence, we need to assure that the selected test data sets represent quite well the variety of the complete set. Likewise, if no stable parameter range for this test set can be found, we can conclude that it is impossible to use the automatic segmentation algorithm with one parameter setting for all datasets.

To find suitable parameters, we created sets of microtubule centerlines manually (the *ground truth*) for the test data and compare the automatically computed centerlines to these. The measurement we use is similar to the one utilized by Cole et al. [15] and results in numerical measures Precision and Recall which measure the estimated error rate in the algorithmic segmentation. Note that using Precision and Recall as measures requires that the ground truth segmentation is complete, since missing lines would result in a low Precision even for a perfect automatic segmentation. Previous work [64], albeit from a different domain, leads us to believe that setting the Precision in our case to above 0.9 and Recall to above 0.8 will yield trustworthy results.

Tuning parameters according to these constraints is time consuming, since ST, OW, and MC depend on each other. Before Tuner, we investigated the effects of the various parameter settings by fixing two of the three parameters and then varying the third. We then plotted Recall over Precision [15]. In the resulting plot the best parameter choice can be found where the curve is closest to one for both Precision and Recall. A stable range of parameters can be found when plotting Precision/Recall over the parameter, but only with the other parameters fixed. Because of the interdependence of the parameters, this tuning is an iterative process. Tuning by hand can take days to weeks. Adding another model parameter to the tracing increases the complexity even further, since checking the effect of the new parameter would require to compare its effect against all three other parameters.

Using Tuner, we estimated our parameters independently on three different data sets with varying acquisition quality (referred to as $DTHQ$, $DTMQ$ and $STLQ$). For each data

set we started with 50 samples on a Latin Hypercube within our three-dimensional parameter space. The number of samples was constrained by the fact that each segmentation took on average ten minutes. Hence, 50 samples took almost 9 hours to compute.

Using the Pareto Panel we first navigated to a sample point near or in the quality constraint. We filtered out all values for Precision and Recall below the constraint using the interactive colormap and drew a box marking ranges of parameters that lay within the unfiltered area, thus fulfilling the constraint. In order to mark out the full region, we iteratively moved the centre point of the found box to points nearby to search for areas where a larger box could be drawn and resized the box. We refined with 20 new sample points. This procedure was repeated for the two other test data.

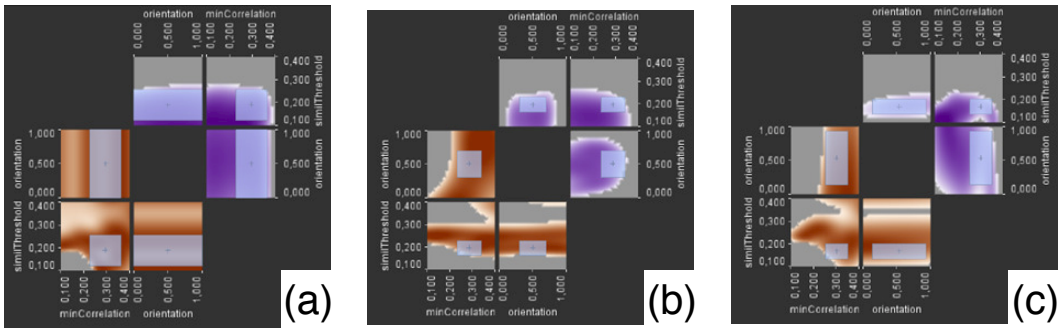


Figure 2.12: Final parameter ranges for the three test datasets. Gray boxes enclose the areas chosen as final ranges for minCorrelation (MC), similThrehold (ST) and orientation (OW) fulfilling both Recall and Precision constraint for a) $DTHQ$, b) $DTMQ$ and c) $STLQ$.

Table 2.1: Resulting parameter ranges for the microtubule segmentation algorithm satisfying the quality constraint.

	min MC	max MC	min ST	max ST	min OW	max OW
DTHQ	0.256	0.37	0.168	0.28	0	1
DTMQ	0.26	0.348	0.197	0.247	0.307	0.702
STLQ	0.278	0.36	0.179	0.235	0.142	0.93
min/max	0.278	0.348	0.197	0.235	0.307	0.702

For each of the three data sets we again searched for valid ranges in the same manner using the refined samples, but this time precisely fulfilling the constraint. Fig. 2.12 shows the resulting choices for each of the data sets. Grey boxes mark the finally chosen ranges. Table 2.1 lists the final ranges for each parameter and data set. A parameter set that fulfills

the constraint for all the datasets must lie inside the intersection of these ranges. The intersection is given in the last line of Table 2.1.

The resulting parameter choices correspond to what we measured by using the above described Precision/Recall plots. However, it was an order of magnitude faster. Moving the slider of one parameter can be considered the same operation as creating a Precision/Recall plot as mentioned above. With Tuner, instead of having to recompute additional values, a simple move of the slider is sufficient. This reduced the work of days to a couple of hours.

Chapter 3

Rendering complexity

In this chapter we analyze the rendering complexity of Tuner's HyperSlice [58] view.

3.1 Motivation

Many scientific studies investigate the relationship between several explanatory variables (input) and a system response variable (output), thereby leading to multi-dimensional data sets. Such data can result from explorations of the parameter space of an image segmentation algorithm where the inputs control aspects of the segmentation like regularization parameter. In this case, the output is actually a segmented image but these segmented images can be scored against a ground truth image to produce numerical output. Another possible source are computer simulations, possibly comparing runs under different model parameter configurations. The common model representation of all these data sets is a continuous input domain and ranges, or outputs, which have been sampled.

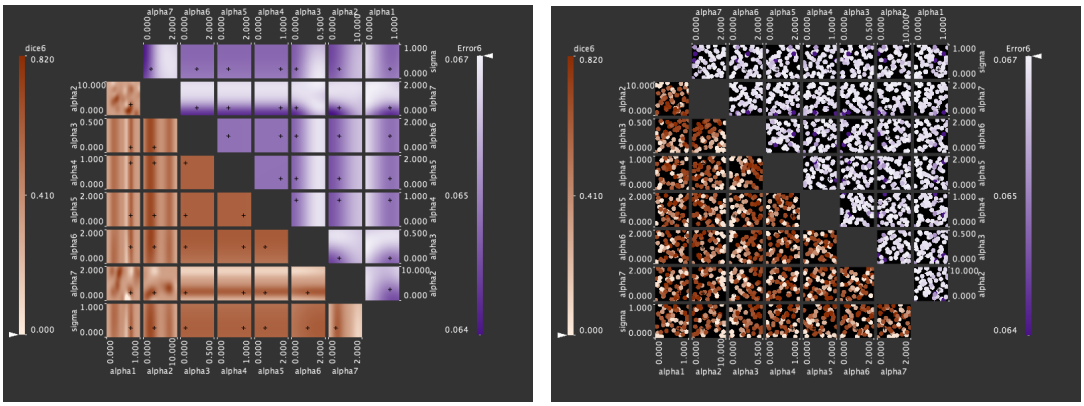
A key step towards learning about mechanisms that are present in a computational model or laws that govern natural phenomena is to study how changes in the input variables affect the output variable. There are numerical methods for investigating this. Local derivative computation and sensitivity indices are two such approaches towards this goal. However, these derived computations have to be set up carefully to yield meaningful results. Multi-dimensional visualization techniques provide a more versatile approach and enable direct human inspection of relationships among variables or data dimensions.

Two common interaction methods for this direct inspection are filtering points using a range query [52], as in a filtered scatterplot, and changing the location of the view or

focus point, as in the HyperSlice [58] method. Both these methods involve interactively changing the filters or focus points to comprehend the data. For the visualization to be effective it is important to maintain interactive frame rates so that we do not cause a cognitive disconnect [51]. Arguments about what exact response time makes a visualization *interactive* vary. However, somewhere between a minimum of 10fps to 60fps are typically deemed acceptable.

In order to prevent this disconnect it would be ideal to know if the number of points in the dataset or the dimensionality of data will overwhelm the graphics capabilities of the user's machine and produce choppy results. Hence, the main aim is predicting the rendering time of a multi-dimensional visualization system. This will guide the user in understanding the complexity and the need to choose the number of sample (data) points carefully, either by offering to sub-sample the dataset, or by guiding the range query.

In order to be able to *predict* the rendering time we must come up with a predictive function for the rendering time based on the size of the multi-dimensional dataset. This function must be adapted to each user's hardware platform so we require a universal methodology that can be run on each user's environment to make accurate predictions.



(a) (b)

Figure 3.1: The two available rendering methods available in Tuner. (a) shows the HyperSlice method, while (b) shows the filtered scatterplot.

We select two rendering methods used in Tuner: a filtered scatterplot, representing interactive filtering, and the HyperSlice method with Gaussian kernel regression, representing interactive viewpoint selection. These are shown in Fig. 3.1. We then describe a methodology to derive and fit a function of the dimensionality, d, the number of data points, N, and

the local search distance, r, to predict the total rendering time for each of the two methods.

The contributions of this analysis are:

- We present a data-driven methodology to predict rendering time for two multi-dimensional scalar field visualization techniques;
- We evaluate this formula against unseen data and a separate architecture to validate its correctness;
- We give a theoretical analysis of the rendering complexity;
- We study this prediction formula to demonstrate implications for the practitioner.

3.2 Rendering algorithm

We consider two visualization methods that produce localized views of multi-dimensional data. HyperSlice is a representative of a continuous reconstruction of the data for rendering, whereas a filtered SPloM shows the data as points. The data is given as a finite set X of $N = |X|$ points inside the d-dimensional unit cube $C = [0, 1]^d$.

Both algorithms are based on a table or matrix arrangement of $\binom{d}{2}$ axis aligned slice views through the data cube C. The bi-variate views in each row/column map the same data dimension (or variable) to their vertical/horizontal axis, respectively. Whether the visualization is based on discrete views — as in the filtered SPloM provided by the Prosection Matrix [57] (*filtered scatterplot* in the following) or continuous methods (HyperSlice [58]) — only the data points that impact this local view actually need to be rendered to the screen. This amounts to a filtering or selection of visible points around the axis aligned slices intersecting at the focus position $\vec{v} \in C$. The viewing region $V(\vec{v})$ around all slices can be used as a visibility test for each data point, yielding a subset of $N' = |X \cap V(\vec{v})|$ visible points. The shape of V differs depending on the method and will receive more discussion in Sec. 3.3.3. This filter-view abstraction leads to a simple algorithm for both rendering methods:

Implementation:

In order to be able to process millions of points in real-time, we will have to map this simple algorithm onto the GPU. However, there are several design decisions to consider.

Algorithm 1 High-level rendering algorithm

Input: viewpoint $\vec{v}$
Output: $\binom{d}{2}$ slices or slabs
1: Determine all visible points $X' = V(\vec{v}) \cap X$
2: Render each of the $\binom{d}{2}$ bi-variate views

For our basic algorithm (Algorithm 1) we need to recompute the set of all visible points $V \cap X$ each time we change the viewpoint $\vec{v}$. This can either be done on the GPU or on the CPU. In order to leverage the extreme degree of parallelism offered by the GPU we use its programmable vertex shader to perform the filtering.

If we render a filtered scatterplot, we simply have to ignore all coordinates of the d-dimensional points, that are not part of the relevant 2D subplot and compute the appropriate screen transform for these points. In the case of a HyperSlice subplot, we will need to compute the distance of the d-dimensional point to the current 2D slice. If this distance is smaller than the size of the radial reconstruction kernel, we would need to render a 2D slice through this reconstruction kernel. Borrowing an idea from GPU-based splatting algorithms [39], we store a template exponential distribution in the GPU texture, that is then projected onto a quad. However, since these splat-like slices have different distances from each subplot they affect the subplot by different amounts depending on the distance. Therefore, we need to scale the intensity value of the texture by its distance to the slice. This leads to the more detailed algorithm:

In our vertex shader implementation we filter the points as well as compute the sliced splat size. Each d-dimensional point is stored in a texture, where the N rows are all the points and the d columns are the coordinates. For each of the $\binom{d}{2}$ subplots we issue a draw command, that draws all N' visible points.

One of the main bottlenecks in the rendering pipeline is transferring vertex data from CPU memory to GPU memory. This is due to the vastly slower speed of the bus compared to the GPU. We simply do not want the GPU waiting for pixel data. The best way to address this, as specified by Xue and Crawfis [62], is to store vertex data in display lists on the GPU. Then, before calling a draw command, we transfer all data needed to render the group of slices to the GPU. Hence, we store all N d-dimensional data points on the GPU. Memory of current GPUs is now large enough that we can easily store millions of points in 10 dimensions directly on the card. Storage does not tend to be the bottleneck of our

Algorithm 2 The full algorithm

Input: viewpoint $\vec{v}$, maximum distance r
Output: $\binom{d}{2}$ slices or slabs

1: **for all** $\binom{d}{2}$ subplots S_i **do** *# filtering*
2: **for all** points $p \in N$ in the texture buffer **do**
3: extract the 2D point location p_{2D} within S_i
4: **for all** remaining $d-2$ dimensions p_{d-2} **do**
5: dist $\leftarrow$ dist(p_{d-2}, S_i)
6: **if** dist $< r$ **then** *# rendering*
7: transform p_{2D} into screen coordinates $\hat{p}_{2D}$
8: compute the splat width $w = \sqrt{r^2 - \text{dist}^2}$
9: send 2D points $(\hat{p}_{2D}(x) \pm w, \hat{p}_{2D}(y) \pm w)$ to the fragment shader
10: **else**
11: send a degenerate triangle to the fragment shader
12: **end if**
13: **end for**
14: **end for**
15: **end for**

multi-dimensional data analysis.

In the case we draw a filtered scatterplot, the distance measure used in step 5 of Algorithm 2 above simply computes the maximum distance over all the dimensions. Further, in this case, we do not have to perform any texture mapping (steps 8 and 9), since we draw a single point.

In the case we draw the HyperSlice matrix, for each point, we create a vertex buffer list, that consists of two integers – one for the point index into the texture, and one integer specifying the corner of the quad to be drawn. Hence, for each quad we issue four points, each with the same point identifier. While we have tried to only issue one integer for one quad and to create the four vertices needed for a quad in the geometry shader, we saw its performance decrease.

3.3 Complexity analysis

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. Our complexity analysis is based on the fact that both our rendering algorithms can be decomposed into a pipeline with filtering and drawing steps. We also assume that our points are uniformly distributed in our data space as this is

how Tuner samples the parameter space. While our regression technique is primarily data-driven having a strong theoretical base for the rendering behaviour allows our model to be more general as well as letting us assign interpretable meaning to the various regression parameters. The rendering algorithm we propose consists of two separate stages. The first stage is a filtering step where we determine which points in space will affect which plots. The second stage draws the points either as point primitives or as quads. Because resources on the GPU are shared between these filtering and drawing stages, the total rendering time, t_{total}, is the greater of the time for the two stages

$$t_{\text{total}} = \max\left(t_{\text{filter}}, t_{\text{render}}\right) \tag{3.1}$$

3.3.1 Filtering

In the filtering step (steps 5 and 6 of Algorithm 2) we must 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, we compute the distance from the sample point to the slice. If the distance is less than r we must draw it.

Since we have subplots for each pair of dimensions there are $\binom{d}{2}$ subplots in total. Therefore, given our architecture-dependent time to filter a single point against a single plot, t_f, the total filtering time, t_{filter}, is

$$t_{\text{filter}} = t_f \binom{d}{2} N \tag{3.2}$$

3.3.2 Rendering

During the rendering step, we only need to process a fraction of the points N, that are visible. We call this fraction N'. In the case of HyperSlice the rendering time is significant. Besides having to determine the size of the quad to be rendered in lines 8 and 9 of Algorithm 2, the actual rendering time depends on the number of pixels covered by the quad [29]. We show an example of the HyperSlice technique being employed by Tuner [55] for an 8-dimensional example in Fig. 3.2. The varying quad sizes display as darker areas in the figure. Because of this, our formulation for the rendering time in the HyperSlice method must include the quad size for each point rendered, q_i, and the time to render each pixel in a quad, t_{H},

$$t_{\text{render}} = t_{\text{H}} \sum_{i=1}^{N'} q_i. \tag{3.3}$$

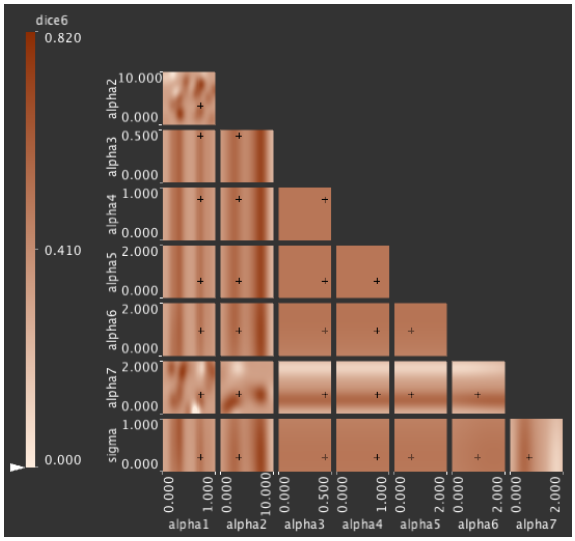


Figure 3.2: Screenshot of Tuner [55] demonstrating the HyperSlice [58] method for rendering an 8-dimensional parameter space using Gaussian kernel reconstruction on an image segmentation dataset. formulation driving the rendering cost for a single slice.

In the case of a filtered scatterplot, we are essentially doing point rendering, and steps 8 and 9 of Algorithm 2 do not need to be executed. Hence, we spend a constant amount of time per point drawn, t_{P}, and our rendering time in this case would amount to

$$t_{\mathrm{render}} = t_{\mathrm{P}} N'. \tag{3.4}$$

Putting (Eq. 3.1) together with (Eq. 3.2), (Eq. 3.3), and (Eq. 3.4) gives us the total rendering time or

$$t^{\mathrm{P}}_{\mathrm{total}} = \max\left(t_f \binom{d}{2} N, t_{\mathrm{P}} N'\right) \tag{3.5}$$

$$t^{\mathrm{H}}_{\mathrm{total}} = \max\left(t_f \binom{d}{2} N, t_{\mathrm{H}} \sum_{i=1}^{N'} q_i\right) \tag{3.6}$$

where $t^{\mathrm{P}}_{\mathrm{total}}$ denotes the complexity of rendering for the filtered scatterplot and $t^{\mathrm{H}}_{\mathrm{total}}$ denotes the complexity of rendering for HyperSlice.

3.3.3 Expected total time

(Eq. 3.5) and (Eq. 3.6) give us the total running time for a particular configuration of N points in d dimensions and for a particular viewpoint $\vec{v}$. However, we are interested in how

well the rendering algorithm performs under many different configurations. Hence, a much more useful measurement is the *average* time to draw the HyperSlice or filtered scatterplot view over all possible point configurations and all possible slice positions. In order to compute this, the expected rendering time $E[t^m_{\text{total}}]$, where $m \in \{\text{H}, \text{P}\}$, is an integral over all point configurations and viewpoints $\vec{v}$ in the unit cube $[0,1]^d$:

$$E[t^m_{total}] = \max\left(t_f \binom{d}{2} N, t_m E_m[N'] E_m[Q]\right) \tag{3.7}$$

where t_m is the time to draw a single fragment using either the HyperSlice method or the scatterplot, $E_m[N']$ is the expected number of points within a distance of r from all 2D slices of the subplot matrix and $E_m[Q]$ is the expected size of a quad drawn. $E_m[N']$ can be further decomposed into $\binom{d}{2} N E_m(r,d)$, where $E_m(r,d)$ is the expected percentage of points within distance r from a single 2D slice in d dimensions. For a uniform distribution of points, this essentially amounts to the volume of a slab of thickness r. The value of $E_m[Q]$ depends on the rendering method used. For the filtered scatterplot method this is a constant as each point drawn results in a dot of constant size. For the HyperSlice technique this quantity depends on the size of the spherical reconstruction kernel which depends on r and d. We denote this quantity $E_{\text{Q}}(r,d)$. It should be pointed out that different ways to measure distance amount to different types of viewing slabs that are of different shape and can greatly vary in volume. A discussion of this effect for differently normed spheres is provided in Bergner [7, § 2.1], where the two examples that apply to our setting are $p = 2$ and $p = \infty$ for the spherical and box kernels respectively. The derivation of the expected count $E_m(r,d)$, performed in Appendix A, considers the volume of the filtering sphere around all points clamped against the bounds of the data set. For the HyperSlice technique we also need to take into account the expected size of the quad in which we are texturing as well.

In summary, these are

$$E_{\mathrm{P}}(r,d) = \left(2r - r^2\right)^{d-2} \tag{3.8}$$

$$\begin{aligned} E_{\mathrm{H}}(r,d) = &\sum_{i=1}^{n-1}(-1)^i\binom{n}{i}\left[\frac{2^{i+1}\pi^{(n-i)/2}r^{n+i}}{\Gamma((n-i)/2)}\prod_{k=0}^{i}\frac{1}{n+i-2k}\right] \\ &+ (-1)^n\frac{r^{2n}}{n!} + \frac{\pi^{\frac{n}{2}}r^n}{\Gamma(\frac{n}{2}+1)} \end{aligned} \tag{3.9}$$

$$\begin{aligned} E_{\mathrm{Q}}(r,d) = \frac{1}{E_{\mathrm{H}}(r,d)}\Bigg(&\sum_{i=1}^{n-1}(-1)^i\binom{n}{i}\frac{4\pi^{(n-i)/2}r^{n+i+2}}{\Gamma\left(\frac{n+i}{2}+2\right)} \\ &- \sum_{i=1}^{n-1}(-1)^i\binom{n}{i}\frac{3\pi^{(n-i+1)/2}r^{n+i+3}}{\Gamma((n+i+5)/2)} \\ &+ \sum_{i=1}^{n-1}(-1)^i\binom{n}{i}\frac{2\pi^{(n-i)/2}r^{n+i+4}}{\Gamma\left(\frac{n+i}{2}+3\right)} \\ &+ \frac{2(-1)^n r^{2n+2}}{(n+1)!} \\ &- \frac{3(-1)^n r^{2n+3}\sqrt{\pi}}{2\Gamma((2n+5)/2)} \\ &+ \frac{(-1)^n r^{2n+4}}{(n+2)!} \\ &+ \frac{4\pi^{n/2}r^{n+2}}{\Gamma(n/2+2)} \\ &- \frac{3\pi^{(n+1)/2}r^{n+3}}{\Gamma((n+5)/2)} \\ &+ \frac{2\pi^{n/2}r^{n+4}}{\Gamma(n/2+3)}\Bigg) \end{aligned} \tag{3.10}$$

where,

$$n = d - 2$$

Here $E_{\mathrm{P}}(r,d)$ is the expected count for the filtered scatterplot method, $E_{\mathrm{H}}(r,d)$ is the expected count for the HyperSlice method, and $E_{\mathrm{Q}}(r,d)$ is the expected quad size given that we need to draw a point for the HyperSlice method. For the filtered scatterplot method $E_{\mathrm{Q}}(r,d) = 1$.

3.4 Calibration

The values t_f, t_{P}, and t_{H} in (Eq. 3.5) and (Eq. 3.6) are dependent on a particular hardware. Hence, we engage in an empirical stage to determine these values for a particular rendering environment.

The architectural design of a GPU is somewhat opaque. Therefore, it is impossible to argue from first-principles how to set t_f, t_{P}, and t_{H} for a particular GPU. We calibrate 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 values we derive are specific to two different architectures in our lab the method we present is applicable elsewhere. We should be able to fit these parameters by properly sampling E and running a regression on the empirical results.

In order to calibrate (Eq. 3.1) we note that the first argument of the max function represents the number of points we need to filter which is constant with respect to the kernel size, r, while the second argument, the drawing time, monotonically 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, that we designate a, at which point the dominant term will change from the first to the second. In order to fit this behaviour we used a piece-wise regression model which changes behaviour according to the value of a $\{0, 1\}$ indicator function $I_a(Q)$ where Q is the total number of fragments drawn. This function returns 0 if $Q < a$ and 1 if $Q > a$. Therefore, we can form the regression formula as:

$$t_{\mathrm{render}} = I_a(Q)\left(N\binom{d}{2}t_f\right) + (1 - I_a(Q))t_m Q \tag{3.11}$$

This formula contains three parameters: t_f (the time to filter one sample point), t_m (the time to draw one fragment), and a (the crossover point). However, since t_{render} must be continuous we can define a as intersection point of the two segments:

$$\begin{aligned} at_m &= N\binom{d}{2}t_f \\ a &= \frac{N\binom{d}{2}t_f}{t_m} \end{aligned} \tag{3.12}$$

This serves to reduce the number of parameters we need to fit to t_f and t_m.

We fit the remaining two parameters using a genetic algorithm optimizing on the mean squared error of prediction against empirical timing results. The issue with using a more common gradient descent method is that the gradient depends on how we set the cutoff point, a.

3.4.1 Sampling E

For each dimension, we would like to ensure, that we have a good spread of values of E. Hence, we must choose different values of r for each d.

Given a desired range of percentages for $E(r, d)$, we can numerically invert this formula, given d and hence, obtain a range of radii r. In order to obtain a sensible range of values for $E(r, d)$ we could choose a particular dimension as a baseline and vary radii in that dimension to come up with a reasonable range of $E(r, d)$.

The last remaining issue is that for each setting of d, N, and r we must generate enough iterations such that the average number of points affecting the slices, N', converges to the theoretical expected value, $E[N']$. We also need to be careful to vary both the distribution of points in the parameter space as well as the viewpoint many times. In our case we found that about 100 frames, where each frame is a change of viewpoint, and 3 sample point distributions for a total of 300 timing measures were necessary for good convergence.

3.5 Results

All tests were performed on two different machine configurations. Our Config A is a CentOS 6 Linux machine using an NVIDIA 8800 GTX GPU with 4GB of RAM and a dual Intel Xeon 3.2 GHz CPU. Config B is a 2012 Apple MacBook Pro "Retina" display with 8GB of RAM, a 2.3 GHz Intel Core i7 CPU, and using an NVIDIA GeForce GT 650M GPU. All code was run using the native OpenGL 3.2 driver and GLSL v1.5 for the shading language.

In our experiments, we constrain our analysis to low-dimensional spaces, i.e. dimensions d between 3 and 8. We determine a reasonable range of percentage values for $E(r, d)$ by computing $E(r, 3)$ in three-dimensional space by changing the spherical radii of influence between 0.05 to 0.5. Further, given a particular d, N, and r, we distribute N points uniformly 3 times and pick 100 different viewpoints v for each point distribution at random.

All timing information is recorded using the internal GPU timer (GL_TIMESTAMP objects) provided by the OpenGL 3.2 spec. The advantage of using this timer over the CPU timer are twofold. This timer has nanosecond resolution so we can see much more fine grained effects on rendering time. This timer also tells us exactly how long the command stream takes to process on the GPU. It avoids any time needed to synchronize the CPU and GPU as well as the effects of any outside processes running on the machine.

3.5.1 Filtered scatterplot

The filtered scatterplot algorithm is characterized as a two-stage process. The filtering step where we decide which data points to draw on which sub-window in the scatterplot matrix and the drawing step where we draw a point on the screen. In order to get the best estimate of both the filtering and drawing time we estimate the filtering time separately.

We began by examining the rendering time as a function of the number of fragments drawn. Since each point in the filtered scatterplot creates a constant number of fragments there is, we would expect, a direct relationship between the number of points drawn and the rendering time. These times are shown in Fig. 3.3. Unfortunately, the MacOSX timer doesn't seem to be as consistent as the Linux timer so there is much more variance in the times. We visually examined Fig. 3.3 which shows that the drawing time has a clear linear dependence with respect to the number of fragments drawn. The parameters for each dimension, d, are recorded in Table 3.1.

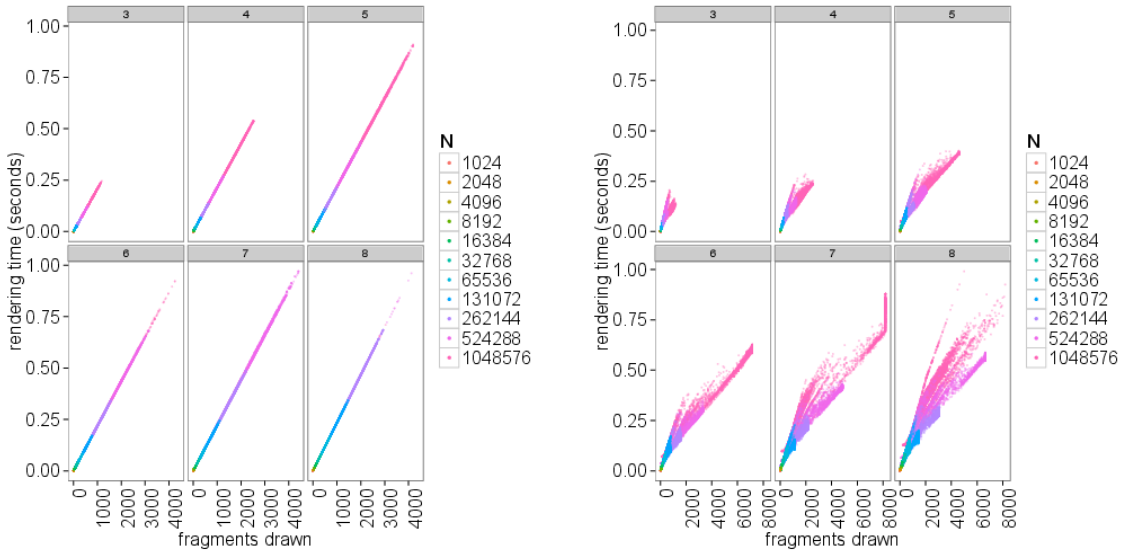
3.5.1.1 Final model

If we then apply these estimates of filter and draw time to (Eq. 3.5) and expand out as in (Eq. 3.11), the full form of our prediction model, conditional on the dimension, d, is

$$t^{\mathrm{P}}_{\mathrm{total}}(d, N, r) = \max\left(t_f(d)\binom{d}{2}N, t_{\mathrm{P}}(d)N'\right) \tag{3.13}$$

$$= I_{a(d)}(Q)\left(t_f(d)\binom{d}{2}N\right) + \left(1 - I_{a(d)}(Q)\right)\left(t_{\mathrm{P}}(d)N'\right) \tag{3.14}$$

where $t_f(d)$, $t_{\mathrm{P}}(d)$, and $I_{a(d)}$ are the parameters we fit, shown in Table 3.1.



(a) Config A

(b) Config B

Figure 3.3: The measured time, in seconds, to render the data points in the filtered scatterplot method. The "spray" in (b) is due to the MacOSX timer not giving as consistent results as the Linux timer.

3.5.2 HyperSlice

Like the filtered scatterplot matrix algorithm, the HyperSlice [58] algorithm is also characterized as a two-stage process with filtering and drawing steps. The filtering step determines which data points to draw on which sub-window in the HyperSlice matrix. Unlike the filtered scatterplot, the drawing step draws a two-dimensional slice of the multidimensional spherical kernel around each data point. Here, the time to render a single data point is much more complex as they are not simply points of a constant size. The size of the data point on the slice reflects its distance from the currently selected focus point. On the GPU, each pixel drawn incurs a cost as we must compute the influence of the Gaussian kernel. Therefore, we must take into account the number of pixels being drawn on screen as well as the number of data points.

We again plot the rendering time as a function of number of fragments drawn in Fig. 3.4 for different values of N, d, and r. What is readily apparent, particularly at higher values of d, is that if the number of pixels being drawn is low enough then the total rendering time is always the same. At a certain point the rendering time switches to a linear function of

Table 3.1: Calibration parameters for the filtered scatterplot rendering method. t_f is the time to filter one data point on one plot, t_P is the time to draw a point in the scatterplot on screen, and a is the number of fragments below which the filtering time dominates rendering and above which the rendering time dominates. In the table we show a/N rather than just a as a varies for each value of N (see (Eq. 3.12)). Since a is dependent on d, N, t_f, and t_P we show a/N rather than a as t_f and t_P are represented in nanoseconds.

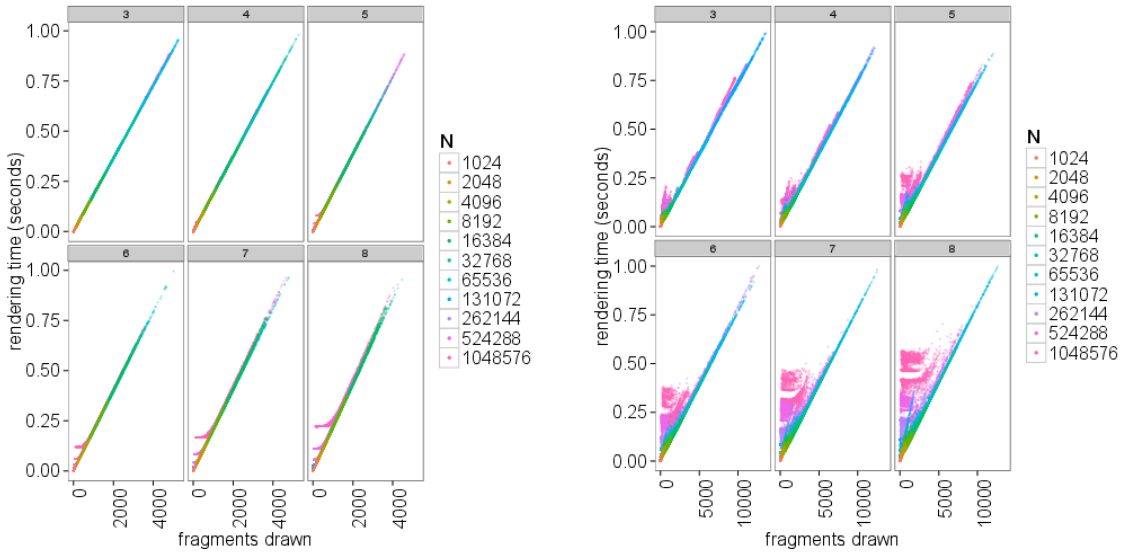
	d	t_f	t_P	a/N
Config A	3	6.02	207888.7	2.90e-5
	4	4.24	212139.7	2.00e-5
	5	2.36	217296.9	1.09e-5
	6	2.51	219502.9	1.14e-5
	7	2.23	221441.1	1.01e-5
	8	1.78	235488.5	7.57e-6
Config B	3	9.94	174246.0	5.71e-5
	4	7.31	116814.3	6.26e-5
	5	6.81	101208.4	6.73e-5
	6	7.29	91795.4	7.95e-5
	7	9.58	99564.7	9.63e-5
	8	9.79	105331.0	9.30e-5

the number of fragments. The flat section represents the filtering time. This is constant because we must always process the N points whether or not they are drawn. At this point the filtering time is so dominant that we do not see any timing effects from drawing. The switch in behaviour occurs when the number of pixels being drawn exceeds the rendering time.

As in the filtered scatterplot method, we fit each dimension separately. In particular, the value of t_f is affected by our distance computation which is implemented using SIMD instructions. On the GPU the largest SIMD size is 4-wide so when we move past 4 dimensions we must use additional SIMD registers in the computation. This affects the value of t_f, particularly as d gets high. The results of our fitting is shown in Table 3.2.

3.5.2.1 Final model

If we then apply these estimates of filter and draw time to (Eq. 3.6) and expand out as in (Eq. 3.11), the full form of our prediction model, conditional on the dimension, d, is



(a) Config A (b) Config B

Figure 3.4: Scatterplots of the time to render using the HyperSlice method. The colours represent different values for the number of sample points. The hockey stick shape is due to whether the filtering time or drawing time is dominant. Again, the "spray" in (b) is due to the MacOSX timer not giving as consistent results as the Linux timer.

$$
\begin{aligned}
t^{\mathrm{H}}_{\mathrm{total}}(d,N,r) &= \max\left(t_f(d)\binom{d}{2}N, t_{\mathrm{H}}(d)E[N']E[Q]\right) \\
&= \max\left(t_f(d)\binom{d}{2}N, t_{\mathrm{H}}(d)E_{\mathrm{H}}(d,r)E_{\mathrm{Q}}(d,r)\right) \\
t^{\mathrm{H}}_{\mathrm{total}}(d,N,r) &= I_{a(d)}(Q)\left(t_f(d)\binom{d}{2}N\right) \\
&\quad + (1 - I_{a(d)}(Q))\left(t_{\mathrm{H}}(d)E_{\mathrm{H}}(d,r)E_{\mathrm{Q}}(d,r)\right) \qquad (3.15)
\end{aligned}
$$

where $t_f(d)$, $t_{\mathrm{H}}(d)$, and $I_{a(d)}$ are the parameters we fit, shown in Table 3.2.

3.6 Accuracy

To test our fitted formula, we compared our predicted running time against new timing data. We tested the effect on rendering time for dimensions ranging from 3 to 8 and points of powers of 2 from 2^{10} to 2^{20}. In order to allow comparison across dimensions we selected

Table 3.2: Calibration parameters for the spherical kernel rendering method. t_f is the time to filter one data point on one plot, t_{H} is the time to draw a sliced point on screen, and a is the number of fragments below which the filtering time dominates rendering and above which the rendering time dominates. In the table we show a/N rather than just a as a varies for each value of N (see (Eq. 3.12)). t_f and t_{H} are represented in nanoseconds.

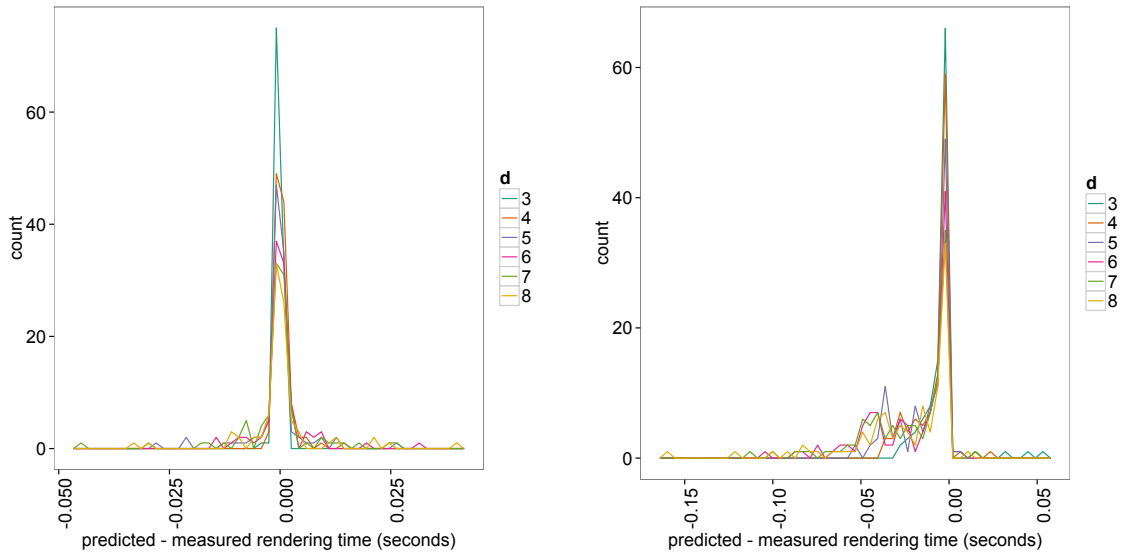
	d	t_f	t_{H}	a/N
Config A	3	9.99	183482.5	5.44e-5
	4	7.62	186951.7	4.08e-5
	5	7.77	194896.2	3.98e-5
	6	7.76	196103.6	3.96e-5
	7	7.74	204933.5	3.78e-5
	8	7.75	220038.7	3.52e-5
Config B	3	9.99	77340.2	0.00013
	4	9.94	76569.5	0.00013
	5	9.98	79230.0	0.00013
	6	9.95	81443.2	0.00012
	7	9.99	84184.4	0.00012
	8	9.86	86870.3	0.00011

r separately for each dimension such that the volume of the d-sphere is constant across dimensions. We used a range of volumes from 0.1 to 1.0 using an increment of 0.1. For each combination of dimension, number of points, and distance threshold we placed the points in the parameter space randomly and uniformly.

We ran each combination three times changing the distribution of points each time. For each distribution of points we allowed the focus point to change by a random direction and amount. The time to render that we report includes the time to first filter which points are required and then to draw the scatterplot points. It is an average value over the number of runs.

3.6.1 Filtered scatterplot

In order to properly evaluate the quality of our fit we examined the residual values between our estimated drawing time and empirical results. We show the absolute as well as relative error in Fig. 3.5 and Fig. 3.6 respectively. We see that overall the prediction is quite good. Furthermore, we are estimating the running time for new data so our estimation is accurate in a purely predictive scenario.



(a) Config A

(b) Config B

Figure 3.5: Histograms showing the predicted value (Eq. 3.13) minus the empirical value for the filtered scatterplot method. The lines are coloured by dimension, d. The empirical data were run using different values for r than used in the training data so the overall error rates here are representative of the predictive ability.

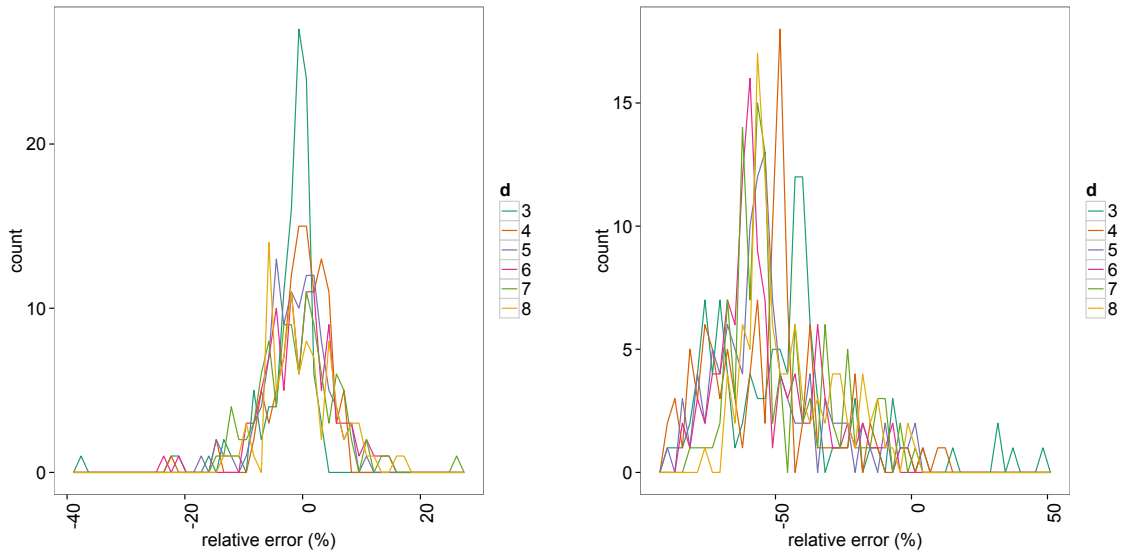
3.6.2 HyperSlice

As with the filtered scatterplot, we compared our predicted running time against new timing data using the same experimental conditions. We do this in order to test new values of r. We then compared the predicted rendering time against the empirically recorded ones. Fig. 3.7 and Fig. 3.8 show the absolute and relative errors for prediction using the HyperSlice method and Gaussian kernel regression respectively.

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, shown in Fig. 3.7, has a strong spike at 0 indicating that most of our predictions are off by a very small amount.

3.7 Application of methodology

While (Eq. 3.13) and (Eq. 3.15) are certainly of theoretical interest, the question we answer in this section is what insights translate into good rules of thumb for a practitioner. Thus,



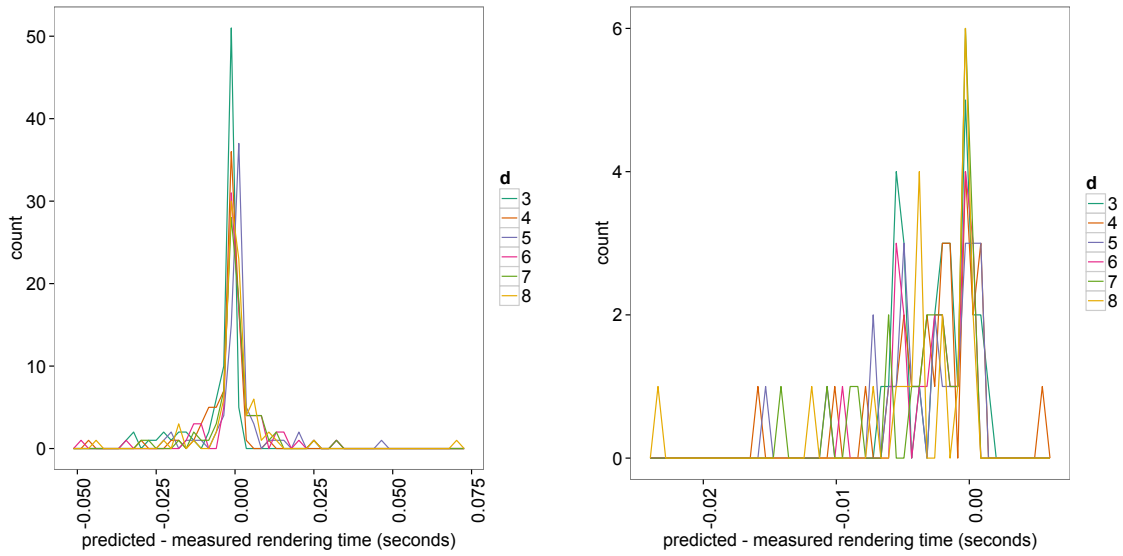
(a) Config A (b) Config B

Figure 3.6: Histograms showing relative error rates for the filtered scatterplot method comparing predictions using (Eq. 3.13) to empirical results. The lines are coloured by dimension, d. The empirical data were run using different values for r than used in the training data so the overall error rates here are representative of the predictive ability.

we present two different applications of these formulas, an examination of how many points we can draw in interactive time as well as a prototypical system that selects sample points based on the desired level of interactivity. All figures in this section use the calibration parameters for Configuration A.

3.7.1 Interactive framerates

It is insightful to study a plot of these two formulas, which can be observed in Fig. 3.9(a) (for a filtered scatterplot – (Eq. 3.13)) and Fig. 3.9(b) (for HyperSlice – (Eq. 3.15)). First, it is typically hard to reason about a particular radius r in higher dimensions. Therefore, in all of the figures in this section, we chose to use the volume of the slice slab as an index instead. In other words, we show all the results for volumes between 0.05 and 1, where a volume of 1 basically means that the slab (filtered scatterplot) or sphere (HyperSlice) has the same size as the unit cube. However, since it doesn't have the same shape (in the case of the spherical kernel) or might not be positioned at the centre of the unit cube, this doesn't



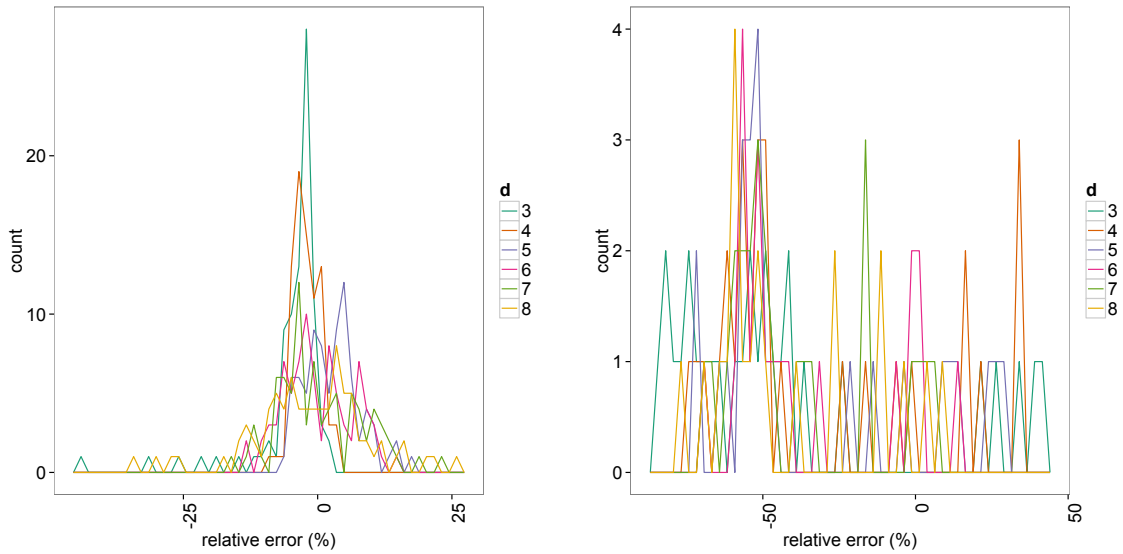
(a) Config A (b) Config B

Figure 3.7: Histograms showing the predicted value (Eq. 3.15) minus the empirical value for the HyperSlice Gaussian kernel technique. The lines are coloured by dimension, d. The empirical data were run using different values for r than used in the training data so the overall error rates here are representative of the predictive ability.

mean that all N points in the unit cube will be visible in that slab or sphere.

For Fig. 3.9(a) and Fig. 3.9(b) we have set $N = 1$, i.e. they are showing the rendering time per point (on average). It is important to note that the rendering time increases with volume. It is worth noting that, while the HyperSlice technique is overall slower, the HyperSlice technique gets faster as d grows while the rendering time of the filtered scatterplot method increases as d increases.

While rendering times are interesting, the more practical question is rather, how many points can be rendered in real-time? For this, we present the inverse of Fig. 3.9(a) in Fig. 3.10(a) and the inverse in Fig. 3.9(b) in Fig. 3.10(b). This is a plot that gives insight into the expected performance of rendering algorithms. As one may expect, we must use fewer sample points as the kernel size increases if we want to stay at the 30fps bound. What is surprising is how much smoother the drop-off is with the spherical kernels (Fig. 3.10(b)) than with the box kernels (Fig. 3.10(a)). There is also an interesting comparison between the volume of the unit box and the unit sphere. As dimensionality d increases, the d-dimensional volume of a d-sphere is generally decreasing while the volume of a unit box stays constant.



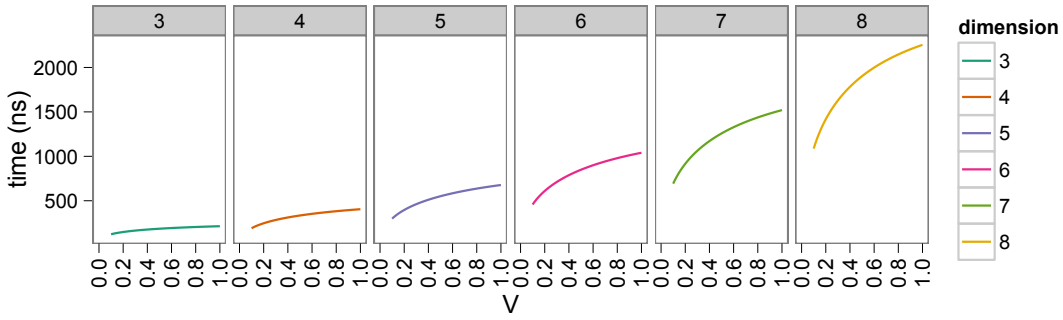
(a) Config A (b) Config B

Figure 3.8: Histograms relative error rates for the HyperSlice Gaussian kernel reconstruction method comparing predictions using (Eq. 3.15) to empirical results. The lines are coloured by dimension, d. The empirical data were run using different values for r than used in the training data so the overall error rates here are representative of the predictive ability.

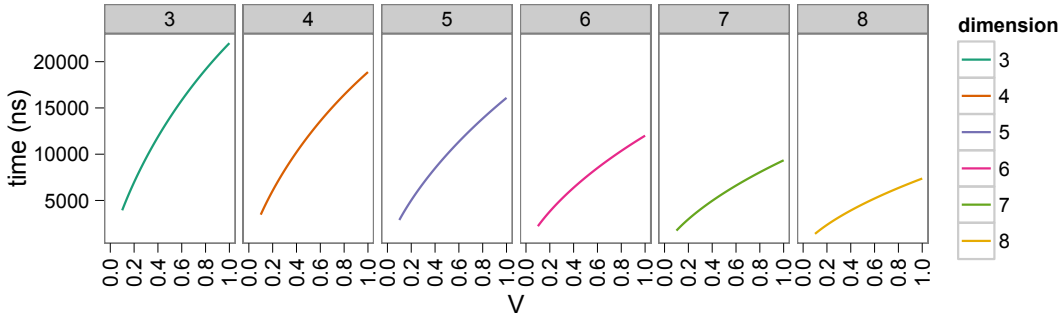
Consequently we can see that in Fig. 3.10(b) the number of points we can render in 30fps actually goes up as dimensionality increases.

3.7.2 Sampling dialog

As was mentioned in Sec. 3.1, there are a number of ways to apply our prediction methodology in a practical visualization system. We show one prototypical scenario in Fig. 3.11. This is a dialog box for the Tuner system, introduced in Chapter 2. The task is to enter the number of sample points to take from the simulation. The dialog is driven by (Eq. 3.13) and (Eq. 3.15). 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 select the number of sample points to take by entering the desired frame rate (B) and letting the system select the number of samples.



(a) Filtered scatterplot



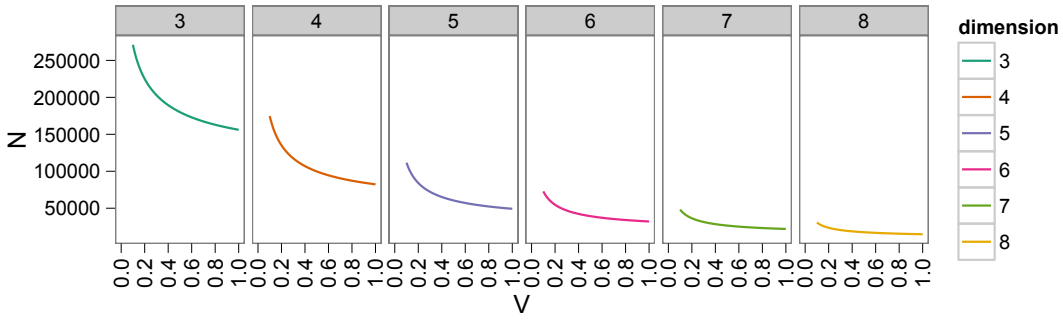
(b) HyperSlice

Figure 3.9: Average render time per sample point (a) for a filtered scatterplot and (b) for the HyperSlice technique.

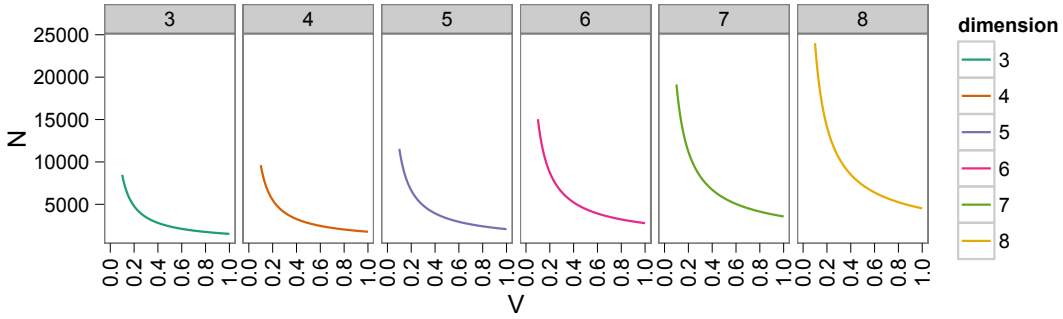
3.7.3 Reconstruction quality

Another important question involves the trade-off between accurately representing our data set and the efficiency of rendering. In terms of accuracy, one may be comfortable if a certain number, k, of the sample points are guaranteed to participate in each sub-plot of the filtered scatterplot. Likewise, for the HyperSlice technique we would like to ensure that a minimum number, k, of points contributes to each pixel of the reconstructed continuous function. If the kernel volume, V, is small then we will need many more sample points in order to get a certain level of k than if V was large. This leads to a formulation for the relationship between the kernel volume (and consequently r), the number of sample points, and k,

$$V = \frac{k}{N}. \tag{3.16}$$



(a) Filtered scatterplot



(b) HyperSlice

Figure 3.10: Average number of points that can be rendered in 30 frames per second (a) for a filtered scatterplot and (b) for the HyperSlice technique.

This is a density measure for the number of sample points appearing on each plot. In Fig. 3.12(a) and Fig. 3.12(b) we show contours for 1, 5, 10, 20, and 30 frames per second. Fig. 3.12(a) shows contours for the filtered scatterplot technique and Fig. 3.12(b) shows contours for the HyperSlice technique. The contours are flat because k is low enough that the filtering time is always dominating the prediction of rendering time. We are simply not drawing enough points on screen for the drawing time to make a difference.

In Fig. 3.13(a) and Fig. 3.13(b) we zoom out to show values of k from 0 to 2000 and again show contours for 1, 5, 10, 20, and 30 frames per second. Here we can see the point where k is large enough that the drawing time begins to come into play. This can be seen in the plots as the point where the slope of the contour line changes. For example, in Fig. 3.13(a), for the 30 frames per second contour we can see the change once $k > 550$. These figures

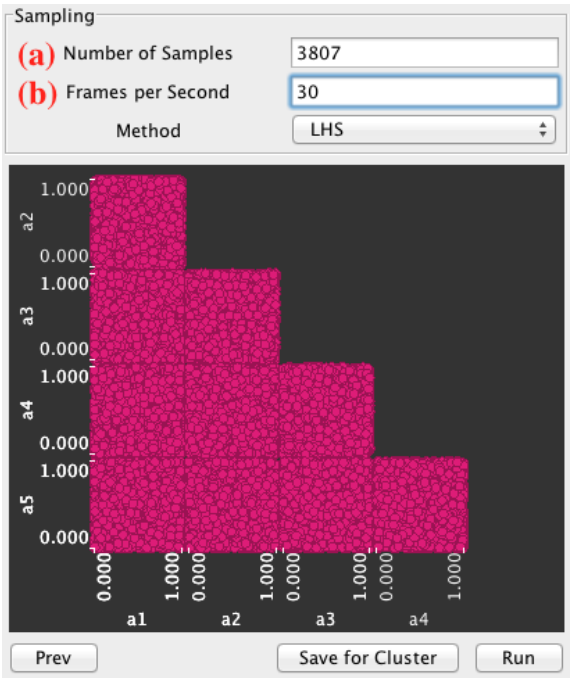
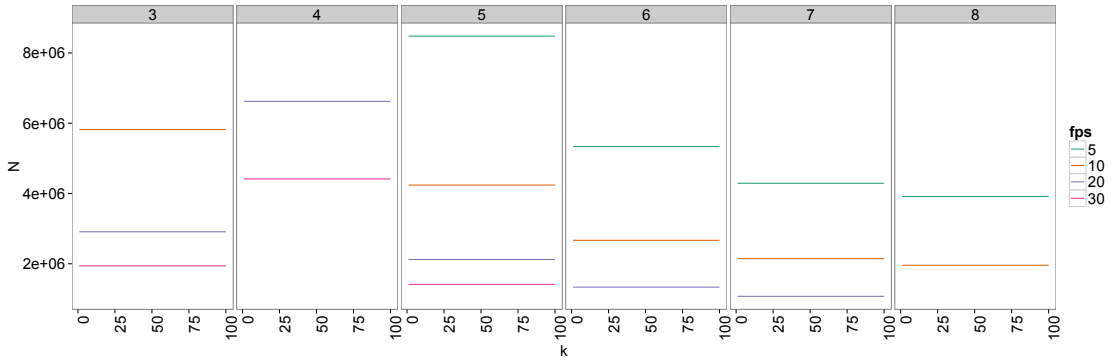
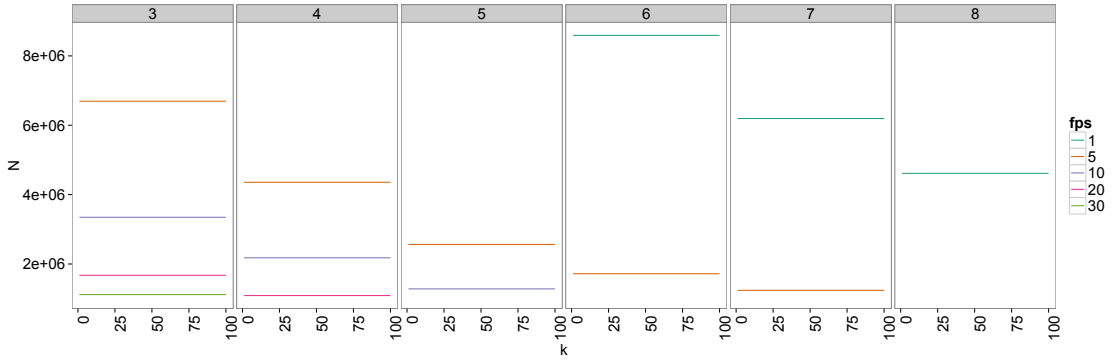


Figure 3.11: A prototypical example use case for our prediction formulas, (Eq. 3.13) and (Eq. 3.15). The user is either able to enter the number of sample points directly in field (a) and the system displays the expected fps in (b) or enter the desired fps directly in (b) and the system calculates the number of points.

also indicate that N is a much stronger determinant of the rendering time than k.

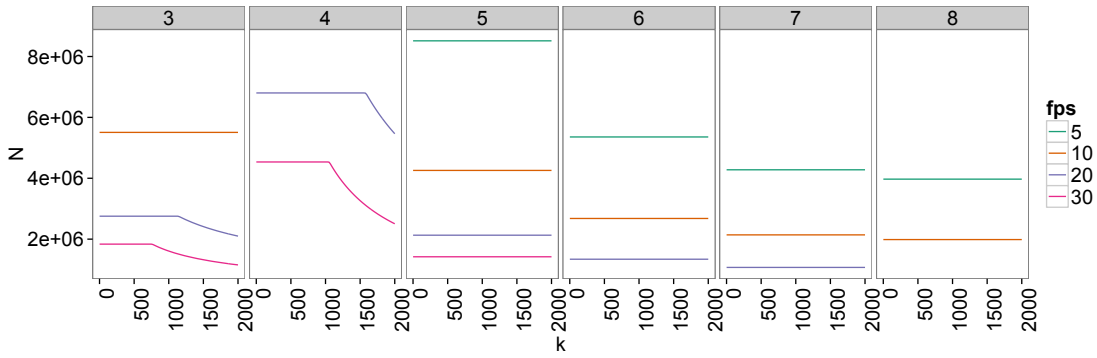


(a) Filtered scatterplot

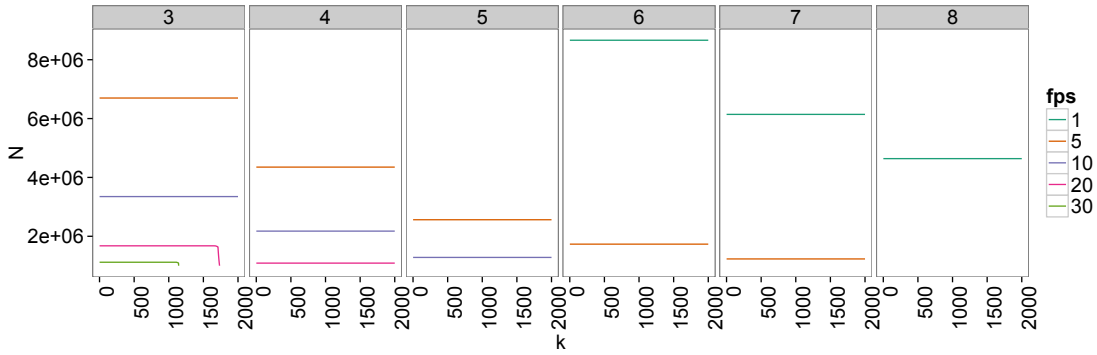


(b) HyperSlice

Figure 3.12: Contour lines showing the trade-off between the number of points used in the reconstruction, k, and the total number of sample points, N, (a) for a filtered scatterplot and (b) for the HyperSlice technique. We show contour lines for frame rates equal to 1, 5, 10, 20, and 30 frames per second. These do not vary with k because it is low enough that the filtering time always dominates the rendering time.



(a) Filtered scatterplot



(b) HyperSlice

Figure 3.13: Contour lines showing the trade-off between the number of points used in the reconstruction, k, and the total number of sample points, N, (a) for a filtered scatterplot and (b) for the HyperSlice technique. We show contour lines for frame rates equal to 1, 5, 10, 20, and 30 frames per second. Note that we require that thousands of points participate in the reconstruction before k begins to be a factor in the rendering time.

Chapter 4

Conclusion

In Chapter 2 we demonstrated Tuner, a tool designed to assist developers of segmentation algorithms with finding "good" parameter settings for a wide variety of algorithms. We began by introducing four tasks that are vital to segmentation algorithm developers: exploring the full parameter space, finding optimal parameter settings, assessing the sensitivity of a parameter region, and simultaneous exploration of multiple quality measures. We demonstrated how our technique of building a statistical model from a sparse sampling, iterative improvement, and high dimensional visualization allow the user to perform these tasks. By using this tool they are able to replace a tedious and manual process with a principled and systematic approach that allows them a much greater understanding of the effect of a parameter on their algorithm. Both our users only spent some hours on the exploration of the parameter space using Tuner before becoming confident on particular optimal parameter regions. Both users had used days and weeks on the same task before Tuner was made available to them.

While the primary users of Tuner to this point have been image segmentation researchers, Tuner itself is a much more general tool capable of guiding exploration and optimization on any application with one or high dimensional scalar fields. A tool like this will only improve the more users we have. Consequently, in the future we hope to deploy it to scientists in a diverse set of fields.

In Chapter 3 we presented a prediction formula capable of accurately estimating the rendering time necessary to render uniformly distributed data in a high-dimensional parameter space. We calibrated this estimation formula for rendering algorithms designed to take advantage of many features of a modern GPU. We then showed two application examples

of this methodology.

We also do not consider the case where the kernels are anisotropic and data are not uniformly distributed. This is in fact the way Tuner's kernels are set up. We would extend our analysis to evaluate these effects.

With this prediction method integrated into Tuner with a suitable subsampling method the application could maintain the important interactive frame rates while still allowing users to see details. In addition, for very local analysis one could do the continuous reconstruction and then switch to a discrete, scatterplot matrix view when the predicted frame rate drops below a threshold. Additionally, with this prediction formula one may actually use the predicted running time as one of the parameters in picking a kernel bandwidth. The effect on the estimation quality of the Gaussian process model is not immediately apparent and warrants further study.

The further development of both these lines of research as well as incorporating additional ideas from sampling theory, multi-objective optimization, and user-centred design will allow us to build a full tool guiding a researcher through the entire analysis pipeline. They will use the computer for repetitive tasks and large-scale computation (which the computer is good at) while the user will be responsible for final decision making and assigning preferences.

Bibliography

[1] Francis J. Anscombe. Graphs in statistical analysis. *The American Statistician*, 27(1):17–21, 1973.

[2] Sven Bachthaler and Daniel Weiskopf. Continuous scatterplots. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1428–1435, December 2008.

[3] Thomas Bartz-Beielstein, Christian W. G. Lasarczyk, and Mike Preuss. Sequential parameter optimization. In *The 2005 IEEE Congress on Evolutionary Computation*, pages 773–780 Vol.1, September 2005.

[4] M. J. Bayarri, James O. Berger, Eliza S. Calder, Keith Dalbey, Simon Lunagomez, Abani K. Patra, E. Bruce Pitman, Elaine T. Spiller, and Robert L. Wolpert. Using statistical and computer models to quantify volcanic hazards. *Technometrics*, 51(4):402–413, November 2009.

[5] Wolfgang Berger and Harald Piringer. Interactive visual analysis of multiobjective optimizations. In *2010 IEEE Symposium on Visual Analytics Science and Technology (VAST)*, pages 215–216, October 2010.

[6] Wolfgang Berger, Harald Piringer, Peter Filzmoser, and Eduard Gröller. Uncertainty-aware exploration of continuous parameter spaces using multivariate prediction. *Computer Graphics Forum (Proceedings of EuroVis 2011)*, 30(3):911–920, June 2011.

[7] Steven Bergner. *Making choices in multi-dimensional parameter spaces.* PhD thesis, Simon Fraser Univsersity, 2011.

[8] George E. P. Box and K. B. Wilson. On the experimental attainment of optimum conditions. *Journal of the Royal Statistical Society. Series B (Methodological)*, 13(1):pp. 1–45, 1951.

[9] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, November 2001.

[10] Eric Brochu, Tyson Brochu, and Nando de Freitas. A Bayesian interactive optimization approach to procedural animation design. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 103–112, 2010.

[11] Stefan Bruckner and Torsten Möller. Result-driven exploration of simulation parameter spaces for visual effects design. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1468–1476, December 2010.

[12] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, November 1986.

[13] 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*, 99(93):919–935, 1994.

[14] William S. Cleveland. *The elements of graphing data*. Belmont, CA, USA, 1985.

[15] Forrester Cole, Aleksey Golovinskiy, Alex Limpaecher, Heather S. Barros, Adam Finkelstein, Thomas Funkhouser, and Szymon Rusinkiewicz. Where do people draw lines? *ACM Transactions on Graphics*, 27(3):107–115, August 2008.

[16] Lee R. Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, July 1945.

[17] Mark Everingham, Henk Muller, and Barry Thomas. Evaluating image segmentation algorithms using the Pareto front. In *Proceedings of the 7th European Conference on Computer Vision*, ECCV 2002, pages 34–48, 2002.

[18] Michael Gordon and Manfred Kochen. Recall-precision trade-off: A derivation. *Journal of the American Society for Information Science*, 40(3):145–151, May 1989.

[19] William A. Gough and William J. Welch. Parameter space exploration of an ocean general circulation model using an isopycnal mixing parameterization. *Journal of Marine Research*, 52(5):773–796, September 1994.

[20] Leo Grady. Random walks for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(11):1768–1783, November 2006.

[21] 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.

[22] Katrin Heitmann, David Higdon, Martin White, Salman Habib, Brian J. Williams, Earl Lawrence, and Christian Wagner. The Coyote Universe II: Cosmological models and precision emulation of the nonlinear matter power spectrum. *The Astrophysical Journal*, 705(1):156–174, November 2009.

[23] Katrin Heitmann, Martin White, Christian Wagner, Salman Habib, and David Higdon. The Coyote Universe I: Precision determination of the nonlinear matter power spectrum. *The Astrophysical Journal*, 715(1):104–121, April 2010.

[24] 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, January 2010.

[25] Alfred Inselberg. The plane with parallel coordinates. *The Visual Computer*, 1:69–91, August 1985.

[26] T. J. Jankun-Kelly and Kwan-Liu Ma. A spreadsheet interface for visualization exploration. In *Proceedings of the 11th IEEE Conference on Visualization (Vis '00)*, pages 69–76, October 2000.

[27] Donald R. Jones, Matthias Schonlau, and William J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4):455–492, December 1998.

[28] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, January 1988.

[29] Steven Kilthau and Torsten Möller. Splatting Optimizations. Technical Report SFU-CMPT-04/01-TR2001-02), School of Computing Science, Simon Fraser University, April 2001.

[30] Sanjiv Kumar and Martial Hebert. Discriminative random fields: A discriminative framework for contextual interaction in classification. In *Proceedings of the Ninth IEEE International Conference on Computer Vision*, pages 1150–1157, October 2003.

[31] Earl Lawrence, Katrin Heitmann, Martin White, David Higdon, Christian Wagner, Salman Habib, and Brian Williams. The Coyote Universe III: Simulation suite and precision emulator for the nonlinear matter power spectrum. *The Astrophysical Journal*, 713(2):1322–1331, April 2010.

[32] Kwan-Liu Ma. Image graphs – a novel approach to visual data exploration. In *Proceedings of IEEE Visualization 1999*, pages 81–513, October 1999.

[33] J. Richard MacIntosh, editor. *Cellular Electron Microscopy.* Avenel, NJ, USA, 2007.

[34] Joe Marks, Brad Andalman, Paul A. Beardsley, William Freeman, Sarah Gibson, Jessica Hodgins, Thomas Kang, Brian Mirtich, Hanspeter Pfister, Wheeler Ruml, Kathy Ryall, Joshua Seims, and Stuart Shieber. Design Galleries: A general approach to setting parameters for computer graphics and animation. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, pages 389–400, 1997.

[35] Chris Mcintosh and Ghassan Hamarneh. Is a single energy functional sufficient? Adaptive energy functionals and automatic initialization. In *Proceedings of the 10th International Conference on Medical Image Computing and Computer-Assisted Intervention*, MICCAI '07, pages 503–510, 2007.

[36] Chris Mcintosh and Ghassan Hamarneh. Optimal weights for convex functionals in medical image segmentation. In *Proceedings of the 5th International Symposium on Advances in Visual Computing*, pages 1079–1088, 2009.

[37] Michael 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, February 1979.

[38] Amar Mitiche and Ismail Ben Ayed. *Variational and Level Set Methods in Image Segmentation*, volume 5 of *Springer Topics in Signal Processing*. 2011.

[39] Klaus Mueller, Torsten Möller, J. Edward Swan II, Roger Crawfis, Naeem Shareef, and Roni Yagel. Splatting errors and antialiasing. In *IEEE Transactions on Visualization and Computer Graphics*, pages 178–191, June 1998.

[40] Jeremy E. Oakley and Anthony O'Hagan. Probabilistic sensitivity analysis of complex models: A Bayesian approach. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 66(3):751–769, August 2004.

[41] Dzung L. Pham, Chenyang Xu, , and Jerry L. Prince. Current methods in medical image segmentation. *Annual Review of Biomedical Engineering*, 2:315–337, August 2000.

[42] Harald Piringer, Wolfgang Berger, and J. Krasser. HyperMoVal: Interactive visual validation of regression models for real-time simulation. *Computer Graphics Forum*, 29(3):983–992, August 2010.

[43] A. Johannes Pretorius, Mark-Anthony P. Bray, Anne E. Carpenter, and Roy A. Ruddle. Visualization of parameter space for image analysis. *IEEE Transactions on Visualization and Computer Graphics*, 17:2402–2411, December 2011.

[44] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning*. 2006.

[45] Ahmed Saad, Ghassan Hamarneh, Torsten Möller, and Ben Smith. Kinetic modeling based probabilistic segmentation for molecular images. In *Proceedings of the 11th International Conference on Medical Image Computing and Computer-Assisted Intervention - Part I*, MICCAI '08, pages 244–252, September 2008.

[46] Jerome Sacks, William J. Welch, Toby J. Mitchell, and Henry P. Wynn. Design and analysis of computer experiments. *Statistical Science*, 4(4):409–423, November 1989.

[47] Thomas J. Santner, Brian J. Williams, and William I. Notz. *The Design and Analysis of Computer Experiments*. 2003.

[48] Carlos Scheidegger, Huy Vo, David Koop, Juliana Freire, and Claudio Silva. Querying and creating visualizations by analogy. *IEEE Transactions on Visualization and Computer Graphics*, 13:1560–1567, November 2007.

[49] Matthias Schonlau and William J. Welch. Screening the input variables to a computer model via analysis of variance and visualization. In Angela Dean and Susan Lewis, editors, *Screening*, pages 308–327. 2006.

[50] Mehmet Sezgin and Blent Sankur. Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic Imaging*, 13(1):146–168, January 2004.

[51] Ben Shneiderman. *Designing the User Interface.* Strategies for Effective Human-Computer Interaction. Reading, MA, 1987.

[52] Ben Shneiderman. Dynamic queries for visual information seeking. *IEEE Software*, 11(6):70–77, November 1994.

[53] Martin Szummer, Pushmeet Kohli, and Derek Hoiem. Learning CRFs using graph cuts. In *Proceedings of the European Conference on Computer Vision*, pages 582–595, October 2008.

[54] Justin Talbot, Sharon Lin, and Pat Hanrahan. An extension of Wilkinson's algorithm for positioning tick labels on axes. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1036–1043, November 2010.

[55] 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(6):1892–1901, August 2011.

[56] 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.

[57] Lisa Tweedie and Robert Spence. The Prosection Matrix: A tool to support the interactive exploration of statistical models and data. *Computational Statistics*, 13(1):65–76, 1998.

[58] Jarke J. van Wijk and Robert van Liere. Hyperslice: Visualization of scalar functions of many variables. In *Proceedings of the 4th Conference on Visualization '93*, pages 119–125, 1993.

[59] Matthew O. Ward. XmdvTool: Integrating multiple methods for visualizing multivariate data. In *Proceedings of the Conference on Visualization '94*, pages 326–333, October 1994.

[60] Jürgen Waser, Raphael Fuchs, Hrvoje Ribicić, Benjamin Schindler, Günther Blöschl, and Eduard Gröller. World lines. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1458–67, December 2010.

[61] Britta Weber, Garrett Greenan, Steffen Prohaska, Daniel Baum, Hans-Christian Hege, Thomas Müller-Reichert, Anthony A. Hyman, and Jean-Marc Verbavatz. Automated tracing of microtubules in electron tomograms of plastic embedded samples of caenorhabditis elegans embryos. *Journal of Structural Biology*, 178(2):129 – 138, December 2012.

[62] Daqing Xue and Roger Crawfis. Efficient splatting using modern graphics hardware. *Journal of Graphics, GPU, and Game Tools*, 8(3):1–21, 2003.

[63] Peng Zhao and Bin Yu. Stagewise Lasso. *Journal of Machine Learning Research*, 8:2701–2726, January 2007.

[64] Yuanxin Zhu, Bridget Carragher, Robert M. Glaeser, Denis Fellmann, Chandrajit Bajaj, Marshall Bern, Fabrice Mouche, Felix de Haas, Richard J. Hall, David Kriegman, Steven J. Ludtke, Satya Mallick, Pawel A. Penczek, Alan M. Roseman, Fred J. Sigworth, Niels Volkmann, and Clinton S. Potter. Automatic particle selection: Results of a comparative study. *Journal of Structural Biology*, 145(1-2):3–14, February 2004.

Appendix A

The expected volume inside a parameter space

A.1 Meaning of $E_m(r, d)$

In this appendix, we are assuming a d-dimensional centred unit cube[1] $[-0.5, 0.5]^d$ with coordinate axis $x_1, x_2, \ldots, x_d$. Without loss of generality, we will specify a 2D slice by a $(d-2)$-dimensional (focus) point $\boldsymbol{s}$ and are assuming, that the additional coordinates $(d-1, d)$ are the unspecified slice coordinates.

As explained in Sec. 3.3.3, $E_m(r, d)$ 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 volume in dimensions $(d-1, d)$ is simply one, since we are examining a d-dimensional unit cube. Hence, the volume of this slab is simply the $(d-2)$-dimensional volume $V(\boldsymbol{s})$ around the $(d-2)$-dimensional point $\boldsymbol{s}$ multiplied by one for each direction $(d-1)$ and d:

$$V(\boldsymbol{s}) = 1 \cdot 1 \cdot \int_{[-0.5,0.5]^{d-2}} B_r(\boldsymbol{s} - \boldsymbol{x}) d\boldsymbol{x} \tag{A.1}$$

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

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

[1] Note, that in Tuner we had specified a non-centred unit cube $[0, 1]^d$. However, the centring doesn't impact the result, but is mathematically easier to deal with.

Here we should note, that the shape of the ball B_r depends on the distance metric we are using. If this distance metric is the Euclidean norm, we essentially are dealing with the HyperSlice technique. If, instead, we are dealing with the infinity-norm, we are dealing with the filtered scatterplot technique.

Considering the $(d-2)$-dimensional centred unit box $\Pi(\boldsymbol{x})$, we can express (Eq. A.1) as a convolution:

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

Since $E_m(r,d)$ is the average volume over all possible slice positions within the $(d-2)$-dimensional unit cube, we conclude:

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

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 these functions over the domain of the third function:

$$E_m(r,d) = \int_{B_r} T(\boldsymbol{x})d\boldsymbol{x} \tag{A.2}$$

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

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

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

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

In the remainder of this appendix we will evaluate the integral in (Eq. A.2) for the two cases of a filtered scatterplot and the HyperSlice techniques. Since this is an integral in a $(d-2)$-dimensional space, we will, for brevity, be using $n = (d-2)$ in the remainder of this appendix.

A.2 The filtered scatterplot

In the case of a filtered scatterplot, the distance metric is an L^∞ metric, i.e. we check each coordinate separately and require that the maximum distance in each component is smaller than our threshold r. This creates an n-dimensional cube of side length $2r$ around the n-dimensional point $\boldsymbol{s}$, specifying the location of our focus point. (Eq. A.2) is now simple to evaluate. Due to the symmetry of the triangle function $T(\boldsymbol{x})$ the integral in each orthant is identical, and we can write:

$$\begin{aligned} E_P(r,d) &= 2^n \int_{B_r^+} T^+(\boldsymbol{x}) d\boldsymbol{x} \\ &= 2^n \int_{B_r^+} \prod_{i=1}^{n} \min(0,(1-x_i)) d\boldsymbol{x} \\ &= 2^n \prod_{i=1}^{n} \int_0^{\min(1,r)} (1-x_i) dx_i \end{aligned}$$

which leads to:

$$E_P(r,d) = \begin{cases} r^n(2-r)^n & \text{if } r < 1 \\ 1 & \text{otherwise} \end{cases}$$

A.3 The HyperSlice case

In the case of the HyperSlice the analysis is very similar, except, that the distance metric is replaced with the L^2 norm and instead the volume B_r is an n-dimensional sphere of radius r. Hence, (Eq. A.2) becomes:

$$E_{\mathrm{H}}(r,d) = 2^n \int_{B_r^+} T^+(\boldsymbol{x}) d\boldsymbol{x} \tag{A.3}$$

We will solve this integral with the help of Polar coordinates.

A.3.1 Polar coordinates in n dimensions

Expressing $\boldsymbol{x} = (x_1, x_2, ..., x_n)$ in polar coordinates $\boldsymbol{\varphi} = (R, \phi_1, \phi_2, \ldots, \phi_{n-1})$ 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 $i = 1, \ldots, n-2$ and $\phi_{n-1} \in [0, 2\pi]$.

The Jacobian, needed to properly substitute the integration variables in (Eq. A.3), of the transformation from spatial coordinates $\boldsymbol{x}$ to polar coordinates $\boldsymbol{\varphi}$ can be computed as follows:

$$\frac{d\boldsymbol{x}}{d\boldsymbol{\varphi}} = R^{n-1}\prod_{i=1}^{n-1}\sin^{n-1-i}(\phi_i) \tag{A.4}$$

$$d\boldsymbol{x} = R^{n-1}dR\prod_{i=1}^{n-1}\sin^{n-1-i}(\phi_i)d\phi_i \tag{A.5}$$

A.3.2 Derivation

In this section, we will assume that our radius r never grows above one. This is a reasonable assumption, which holds for the experiments performed in our paper. We can now simplify (Eq. A.3) as follows:

$$E_{\mathrm{H}}(r, d) = 2^n\int_{B_r^+}\prod_{i=1}^{n}\min(0, (1 - x_i))d\boldsymbol{x} \tag{A.6}$$

$$= 2^n\int_{B_r^+}\prod_{i=1}^{n}(1 - x_i)d\boldsymbol{x} \tag{A.7}$$

$$= 2^n\int_{B_r^+}\sum_{i=0}^{n}(-1)^i t_i(\boldsymbol{x})d\boldsymbol{x} \tag{A.8}$$

$$= 2^n\sum_{i=0}^{n}(-1)^i\int_{B_r^+} t_i(\boldsymbol{x})d\boldsymbol{x} \tag{A.9}$$

where $t_i(\boldsymbol{x})$ is the sum of all products of i distinct coordinates. I.e.

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

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

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

which holds for any product of arbitrary terms. Hence, we can simplify (Eq. A.9) in the following way:

$$\begin{aligned} E_{\mathrm{H}}(r,d) &= 2^n \sum_{i=0}^{n} (-1)^i \int_{B_r^+} t_i(\boldsymbol{x}) d\boldsymbol{x} \\ &= 2^n \sum_{i=1}^{n} (-1)^i \binom{n}{i} \int_{B_r^+} \prod_{k=1}^{i} x_k d\boldsymbol{x} + 2^n \int_{B_r^+} d\boldsymbol{x} \\ &= 2^n \sum_{i=1}^{n} (-1)^i \binom{n}{i} A_i + B \end{aligned}$$

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

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

We are left to compute the product integrals A_i, for which we use polar coordinates. For

$i < n$, we have (using (Eq. A.5)):

$$A_i = \int_{B_r^+} \prod_{k=1}^{i} x_k d\boldsymbol{x} \tag{A.10}$$

$$= \int_{\boldsymbol{\varphi}^+} \prod_{k=1}^{i} x_k(\boldsymbol{\varphi}) R^{n-1} dR \prod_{j=1}^{n-1} \sin^{n-1-j}(\phi_j) d\phi_j \tag{A.11}$$

$$= \int_{\boldsymbol{\varphi}^+} R^i \prod_{k=1}^{i} \cos(\phi_k) \sin^{i-k}(\phi_k) R^{n-1} dR \prod_{j=1}^{n-1} \sin^{n-1-j}(\phi_j) d\phi_j \tag{A.12}$$

$$= \int_{\boldsymbol{\varphi}^+} R^{n+i-1} dR \prod_{k=1}^{i} \cos(\phi_k) \sin^{i-k}(\phi_k) \prod_{j=1}^{n-1} \sin^{n-1-j}(\phi_j) d\phi_j \tag{A.13}$$

$$= \int_{\boldsymbol{\varphi}^+} R^{n+i-1} dR \prod_{k=1}^{i} \cos(\phi_k) \sin^{n-1+i-2k}(\phi_k) \prod_{j=i+1}^{n-1} \sin^{n-1-j}(\phi_j) d\phi_j \tag{A.14}$$

$$= A_i^R \prod_{k=1}^{i} A_{i,k}^c \prod_{j=i+1}^{n-1} A_{i,j}^s \tag{A.15}$$

where

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

$$A_{i,k}^c = \int_0^{\pi/2} \cos(\phi_k) \sin^{n-1+i-2k}(\phi_k)$$

$$A_{i,j}^s = \int_0^{\pi/2} \sin^{n-1-j}(\phi_j) d\phi_j$$

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

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

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

where F_1 is the hypergeometric function. Putting this all together (for $i < n$) into (Eq. A.15), we get:

$$A_i = A^R_i \prod_{k=1}^{i} A^c_{i,k} \prod_{j=i+1}^{n-1} A^s_{i,j} \tag{A.16}$$

$$= \frac{1}{n+i} r^{n+i} \prod_{k=1}^{i} \frac{1}{n+i-2k} \prod_{j=i+1}^{n-1} \frac{\sqrt{\pi}}{2} \frac{\Gamma((n-j)/2)}{\Gamma((n-j+1)/2)} \tag{A.17}$$

$$= r^{n+i} \prod_{k=0}^{i} \frac{1}{n+i-2k} \frac{\sqrt{\pi}^{n-1-i}}{2^{n-1-i}} \frac{\Gamma(1/2)}{\Gamma((n-i)/2)} \tag{A.18}$$

$$= r^{n+i} \frac{\sqrt{\pi}^{n-1-i}}{2^{n-1-i}} \frac{\sqrt{\pi}}{\Gamma((n-i)/2)} \prod_{k=0}^{i} \frac{1}{n+i-2k} \tag{A.19}$$

$$= \frac{\sqrt{\pi}^{n-i}}{2^{n-1-i}} \frac{r^{n+i}}{\Gamma((n-i)/2)} \prod_{k=0}^{i} \frac{1}{n+i-2k} \tag{A.20}$$

Now, solving for A_n, we have:

$$\begin{aligned}
A_n &= \int_{B^+} \prod_{k=1}^{n} x_k d\boldsymbol{x} \\
&= \int_{\boldsymbol{\varphi}^+} \prod_{k=1}^{n} x_k(\boldsymbol{\varphi}) R^{n-1} dR \prod_{j=1}^{n-1} \sin^{n-1-j}(\phi_j) d\phi_j \\
&= \int_{\boldsymbol{\varphi}^+} R^n \prod_{k=1}^{n-1} \cos(\phi_k) \sin^{n-k}(\phi_k) R^{n-1} dR \prod_{j=1}^{n-1} \sin^{n-1-j}(\phi_j) d\phi_j \\
&= \int_{\boldsymbol{\varphi}^+} R^{2n-1} dR \prod_{k=1}^{n-1} \cos(\phi_k) \sin^{n-k}(\phi_k) \prod_{j=1}^{n-1} \sin^{n-1-j}(\phi_j) d\phi_j \\
&= \int_{\boldsymbol{\varphi}^+} R^{2n-1} dR \prod_{k=1}^{n-1} \cos(\phi_k) \sin^{2n-1-2k}(\phi_k) d\phi_k \\
&= \int_{0}^{r} R^{2n-1} dR \prod_{k=1}^{n-1} \int_{0}^{\pi/2} \cos(\phi_k) \sin^{2n-1-2k}(\phi_k) d\phi_k \\
&= A_n^R \prod_{k=1}^{n-1} A_{n,k}^c \\
&= \frac{1}{2n} R^{2n} \prod_{k=1}^{n-1} \frac{1}{2n-2k} \\
&= 2^{-n} R^{2n} \prod_{k=0}^{n-1} \frac{1}{n-k} \\
&= \frac{2^{-n}}{n!} R^{2n}
\end{aligned}$$

Finally, we can put everything together:

$$\begin{aligned}
E_{\mathrm{H}}(r,d) &= 2^n \sum_{i=1}^{n} (-1)^i \binom{n}{i} A_i + B \\
&= 2^n \sum_{i=1}^{n-1} (-1)^i \binom{n}{i} A_i + (-2)^n A_n + B \\
&= 2^n \sum_{i=1}^{n-1} (-1)^i \binom{n}{i} \left[\frac{\sqrt{\pi}^{n-i}}{2^{n-1-i}} \frac{r^{n+i}}{\Gamma((n-i)/2)} \prod_{k=0}^{i} \frac{1}{n+i-2k} \right] + (-2)^n \frac{2^{-n}}{n!} r^{2n} + B \\
&= \sum_{i=1}^{n-1} (-1)^i \binom{n}{i} \left[\frac{2^{i+1} \pi^{(n-i)/2} r^{n+i}}{\Gamma((n-i)/2)} \prod_{k=0}^{i} \frac{1}{n+i-2k} \right] + (-1)^n \frac{r^{2n}}{n!} + \frac{\pi^{\frac{n}{2}} r^n}{\Gamma(\frac{n}{2}+1)}
\end{aligned}$$

A.4 Derivation of $E_{\mathbf{Q}}(r, d)$

With our formulation for the expected number of points appearing on a slice, $E_{\mathrm{H}}(r, d)$, we now turn to the derivation of the expected number of fragments that need to be drawn per point on a slice, $E_{\mathrm{Q}}(r, d)$. We will compute this in two stages. First we derive the expected size 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, we still must process it as a quad since the GPU does not support circle primitives. We then average this value over all possible values of t. For convenience, in the derivation below we drop the functional notation, $E_{\mathrm{H}}(r, d)$ in favour of the more compact $E[q]$.

A.4.1 Expected quad size

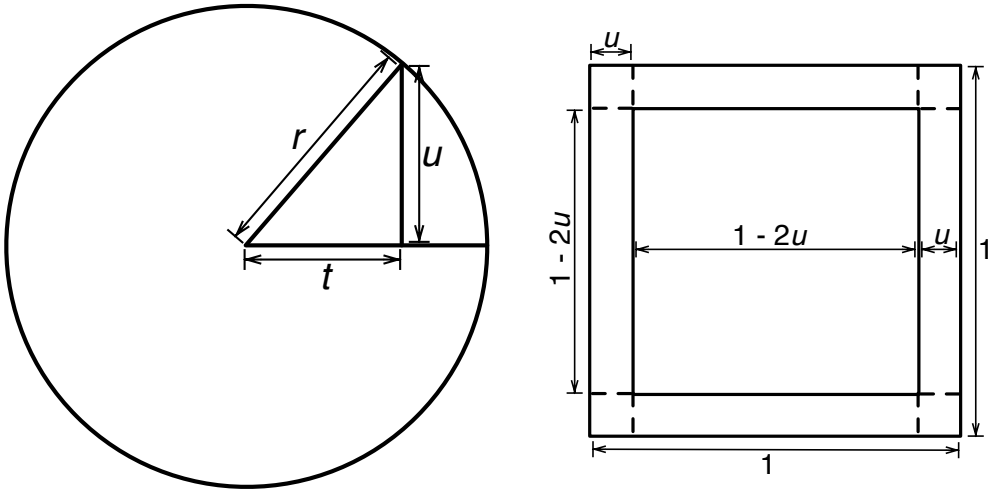
Given that we need to draw a particular data point, the question is how large an impact in terms of number of fragments does it make on the 2D slice. As the distance from a sample point to the slice, t, increases the size of the quad, q, decreases. This is due to the slice passing through a smaller area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

Therefore, u is related to t through $u = \sqrt{r^2 - t^2}$ and the maximum size of the quad is $4u^2$. However, the quad size is not always $4u^2$. If the centre of the quad gets 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 Fig. A.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,

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

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 $P = 4u(1 - 2u)$



(a) (b)

Figure A.1: (a) A 2D cross-section of a hypersphere of radius r representing the spherical kernel centred 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 centre of the sample point lies. If the sample point does not lie in the centre 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.

- **centre:** where the quad size is $4u^2$ with probability $P = (1 - 2u)^2$

Therefore, our 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 centre cases respectively. We solve each integral individually,

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

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

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

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

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

A.4.2 Expected quad size

Now we can turn to a formulation of $E[q]$ which is the expected quad size over all $0 \leq t \leq r$. We also need to take into account the likelihood of a point lying at a particular value of t, which we denote P_t. Here $P_t = \frac{E_{\mathrm{H}}(t,d)}{E_{\mathrm{H}}(r,d)}$.

$$
E[q] = \int_0^r E[q](u) P_t \, dt
$$

There are 2 cases we need to consider, the $d = 3$ case and the $d > 3$ case.

A.4.2.1 $d = 3$ case

When $d = 3$ all points lie on a line extending on either side of the slice. Since all distances are equally likely in this case, we can simply integrate $E[q](u)$ over all t.

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

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

$$\begin{aligned} t &= r \sin \theta \\ dt &= r \cos \theta \, d\theta \\ \sqrt{r^2 - t^2} &= r \cos \theta \end{aligned}$$

which leads to

$$
\begin{aligned}
E[q] &= \int_0^{\pi/2} \left(4(r\cos\,\theta)^2 - 4(r\cos\,\theta)^3 + (r\cos\,\theta)^4\right) r\cos\,\theta\, d\theta \\
&= \int_0^{\pi/2} 4r^3\cos^3\,\theta - 4r^4\cos^4\,\theta + r^5\cos^5\,\theta\, d\theta \\
&= \int_0^{\pi/2} 4r^3\cos^3\,\theta d\theta - \int_0^{\pi/2} 4r^4\cos^4\,\theta d\theta + \int_0^{\pi/2} r^5\cos^5\,\theta\, d\theta \\
&= 4r^3\left(\frac{\cos^2\,\theta\,\sin\,\theta}{3} + \frac{2}{3}\int_0^{\pi/2}\cos\,\theta d\theta\right)\Bigg|_0^{\pi/2} \\
&\quad - 4r^4\left(\frac{\cos^3\,\theta\,\sin\,\theta}{4} + \frac{3}{4}\int_0^{\pi/2}\cos^2\,\theta d\theta\right)\Bigg|_0^{\pi/2} \\
&\quad + r^5\left(\frac{\cos^4\,\theta\,\sin\,\theta}{5} + \frac{4}{5}\int_0^{\pi/2}\cos^3\,\theta d\theta\right)\Bigg|_0^{\pi/2} \\
&= 4r^3\left(\frac{\cos^2\,\theta\,\sin\,\theta}{3} + \frac{2}{3}\sin\,\theta\right)\Bigg|_0^{\pi/2} \\
&\quad - 4r^4\left(\frac{\cos^3\,\theta\,\sin\,\theta}{4} + \frac{3}{4}\left(\frac{\theta}{2} + \frac{1}{4}\sin\,2\theta\right)\right)\Bigg|_0^{\pi/2} \\
&\quad + r^5\left(\frac{\cos^4\,\theta\,\sin\,\theta}{5} + \frac{4}{5}\int_0^{\pi/2}\cos^3\,\theta d\theta\right)\Bigg|_0^{\pi/2} \\
&= 4r^3\left(\frac{2}{3}\right) - 4r^4\left(\frac{3\pi}{16}\right) + r^5\left(\frac{4}{5}\frac{2}{3}\right) \\
E[q] &= \frac{8r^3}{3} - \frac{3\pi r^4}{4} + \frac{8r^5}{15}
\end{aligned}
$$

A.4.2.2 $d > 3$ case

When $d > 3$, the points at distance t lie along the surface of a hyperball. Again, to find $E[q]$ we need to find the average of $E[q](t)$ over all $0 \leq t \leq r$, which means,

$$
\begin{aligned}
E[q] &= \int_0^r E[q](t) P_t \, dt \\
&= \int_0^r \left(4(\sqrt{r^2-t^2})^2 - 4(\sqrt{r^2-t^2})^3 + (\sqrt{r^2-t^2})^4\right) \frac{E_{\mathrm{H}}(t,d)}{E_{\mathrm{H}}(r,d)} \, dt \\
&= \frac{1}{E_{\mathrm{H}}(r,d)} \int_0^r \left(4(\sqrt{r^2-t^2})^2 - 4(\sqrt{r^2-t^2})^3 + (\sqrt{r^2-t^2})^4\right) E_H(t,d) \, dt \\
&= \frac{1}{E_{\mathrm{H}}(r,d)} \int_0^r \left(4(\sqrt{r^2-t^2})^2 - 4(\sqrt{r^2-t^2})^3 + (\sqrt{r^2-t^2})^4\right) \\
&\qquad \left(\sum_{i=1}^{n-1} (-1)^i \binom{n}{i} \frac{2^{i+1} t^{n+i-1} \pi^{(n-i)/2}}{\Gamma((n-i)/2)} \prod_{k=1}^{i} \frac{1}{n+i-2k} \right. \\
&\qquad\qquad \left. + \frac{(-1)^n t^{2n-1}}{(n-1)!} + \frac{n\pi^{n/2} t^{n-1}}{\Gamma(n/2+1)}\right) dt
\end{aligned}
$$

Just to simplify the writing a bit we can replace the non-t parts of the formula by

$$
\begin{aligned}
K_i &= (-1)^i \binom{n}{i} \frac{2^{i+1} \pi^{(n-i)/2}}{\Gamma((n-i)/2)} \prod_{k=1}^{i} \frac{1}{n+i-2k} \\
K_n &= \frac{(-1)^n}{(n-1)!} \\
C_n &= \frac{\pi^{n/2}}{\Gamma(n/2+1)}
\end{aligned}
$$

Then we get,

$$
\begin{aligned}
E[q] = \frac{1}{E_{\mathrm{H}}(r,d)} \int_0^r &\left(4(\sqrt{r^2-t^2})^2 - 4(\sqrt{r^2-t^2})^3 + (\sqrt{r^2-t^2})^4\right) \\
&\left(\sum_{i=1}^{n-1} K_i t^{n+i-1} + K_n t^{2n-1} + n C_n t^{n-1}\right) dt
\end{aligned}
$$

We can also remove the square roots by substituting in for t,

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

Then we get,

$$
\begin{aligned}
&= \frac{1}{E_{\mathrm{H}}(r,d)} \int_0^{\pi/2} \left(4(r\cos(\theta))^2 - 4(r\cos(\theta))^3 + (r\cos(\theta))^4\right) \\
&\qquad \left(\sum_{i=1}^{n-1} K_i (r\sin(\theta))^{n+i-1} \right. \\
&\qquad\quad + K_n (r\sin(\theta))^{2n-1} \\
&\qquad\quad \left. + \, nC_n (r\sin(\theta))^{n-1}\right) r\cos(\theta)\, d\theta \\
&= \frac{1}{E_{\mathrm{H}}(r,d)} \int_0^{\pi/2} \left(4r^2\cos^2(\theta) - 4r^3\cos^3(\theta) + r^4\cos^4(\theta)\right) \\
&\qquad \left(\sum_{i=1}^{n-1} K_i r^{n+i-1} \sin^{n+i-1}(\theta) \right. \\
&\qquad\quad + K_n r^{2n-1} \sin^{2n-1}(\theta) \\
&\qquad\quad \left. + \, nC_n r^{n-1} \sin^{n-1}(\theta)\right) r\cos(\theta)\, d\theta \\
&= \frac{1}{E_{\mathrm{H}}(r,d)} \int_0^{\pi/2} \left(4r^3\cos^3(\theta) - 4r^4\cos^4(\theta) + r^5\cos^5(\theta)\right) \\
&\qquad \left(\sum_{i=1}^{n-1} K_i r^{n+i-1} \sin^{n+i-1}(\theta) \right. \\
&\qquad\quad + K_n r^{2n-1} \sin^{2n-1}(\theta) \\
&\qquad\quad \left. + \, nC_n r^{n-1} \sin^{n-1}(\theta)\right)\, d\theta
\end{aligned}
$$

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

$$
\begin{aligned}
= \frac{1}{E_{\mathrm{H}}(r,d)} \Bigg(& \sum_{i=1}^{n-1} 4K_i r^{n+i+2} \int_0^{\pi/2} \cos^3(\theta) \sin^{n+i-1}(\theta)\, d\theta \\
& - \sum_{i=1}^{n-1} 4K_i r^{n+i+3} \int_0^{\pi/2} \cos^4(\theta) \sin^{n+i-1}(\theta)\, d\theta \\
& + \sum_{i=1}^{n-1} K_i r^{n+i+4} \int_0^{\pi/2} \cos^5(\theta) \sin^{n+i-1}(\theta)\, d\theta \\
& + 4K_n r^{2n+2} \int_0^{\pi/2} \cos^3(\theta) \sin^{2n-1}(\theta)\, d\theta \\
& - 4K_n r^{2n+3} \int_0^{\pi/2} \cos^4(\theta) \sin^{2n-1}(\theta)\, d\theta \\
& + K_n r^{2n+4} \int_0^{\pi/2} \cos^5(\theta) \sin^{2n-1}(\theta)\, d\theta \\
& + 4nC_n r^{n+2} \int_0^{\pi/2} \cos^3(\theta) \sin^{n-1}(\theta)\, d\theta \\
& - 4nC_n r^{n+3} \int_0^{\pi/2} \cos^4(\theta) \sin^{n-1}(\theta)\, d\theta \\
& + nC_n r^{n+4} \int_0^{\pi/2} \cos^5(\theta) \sin^{n-1}(\theta)\, d\theta \Bigg)
\end{aligned}
$$

Here note that we can apply the following trigonometric identities:

$$
\begin{aligned}
\int \cos^n(\theta)\, \sin^m(\theta)\, d\theta &= \frac{\sin^{m+1}(\theta)\, \cos^{n-1}(\theta)}{m+n} + \frac{n-1}{m+n} \int \cos^{n-2}(\theta)\, \sin^m(\theta)\, d\theta \\
\int_0^{\pi/2} \sin^m(\theta)\, d\theta &= -\cos(\theta)\, F_1\left(\frac{1}{2}, \frac{1-m}{2}, \frac{3}{2}, \cos^2(\theta)\right)\Bigg|_0^{\pi/2} \\
&= 0 + F_1\left(\frac{1}{2}, \frac{1-m}{2}, \frac{3}{2}, 1\right) \\
&= \frac{\sqrt{\pi}\Gamma\left(\frac{m+1}{2}\right)}{2\Gamma\left(\frac{m}{2}+1\right)}
\end{aligned}
$$

There are really 3 different types of integrals representing the power on the $\cos(\theta)$ term. They are,

$$
\begin{aligned}
I_1(m) &= \int_0^{\pi/2} \cos^3(\theta)\sin^m(\theta)\,d\theta \\
&= \left.\frac{\sin^{m+1}(\theta)\cos^2(\theta)}{m+3}\right|_0^{\pi/2} + \frac{2}{m+3}\int_0^{\pi/2}\cos(\theta)\sin^m(\theta)\,d\theta \\
&= 0 + \frac{2}{m+3}\left.\frac{\sin^{m+1}(\theta)}{m+1}\right|_0^{\pi/2} \\
&= \frac{2}{m+3}\left(\frac{1}{m+1} - 0\right) \\
I_1(m) &= \frac{2}{(m+1)(m+3)} \\
I_2(m) &= \int_0^{\pi/2} \cos^4(\theta)\sin^m(\theta)\,d\theta \\
&= \left.\frac{\sin^{m+1}(\theta)\cos^3(\theta)}{m+4}\right|_0^{\pi/2} + \frac{3}{m+4}\int_0^{\pi/2}\cos^2(\theta)\sin^m(\theta)\,d\theta \\
&= 0 + \frac{3}{m+4}\left[\left.\frac{\sin^{m+1}(\theta)\cos(\theta)}{m+2}\right|_0^{\pi/2} + \frac{1}{m+2}\int_0^{\pi/2}\sin^m(\theta)\,d\theta\right] \\
&= \frac{3}{m+4}\left[0 + \frac{1}{m+2}\frac{\sqrt{\pi}\Gamma((m+1)/2)}{2\Gamma(m/2+1)}\right] \\
I_2(m) &= \frac{3\sqrt{\pi}}{2(m+2)(m+4)}\frac{\Gamma((m+1)/2)}{\Gamma(m/2+1)} \\
I_3(m) &= \int_0^{\pi/2} \cos^5(\theta)\sin^m(\theta)\,d\theta \\
&= \left.\frac{\sin^{m+1}(\theta)\cos^4(\theta)}{m+5}\right|_0^{\pi/2} + \frac{4}{m+5}\int_0^{\pi/2}\cos^3(\theta)\sin^m(\theta)\,d\theta \\
&= 0 + \frac{4}{m+5}\int_0^{\pi/2}\cos^3(\theta)\sin^m(\theta)\,d\theta \\
&= \frac{4}{m+5}I_1(m) \\
&= \frac{4}{m+5}\frac{2}{(m+1)(m+3)} \\
I_3(m) &= \frac{8}{(m+1)(m+3)(m+5)}
\end{aligned}
$$

We can substitute these in,

$$
\begin{aligned}
&= \frac{1}{E_{\mathrm{H}}(r,d)} \Bigg(\sum_{i=1}^{n-1} 4K_i r^{n+i+2} I_1(n+i-1) \\
&\qquad - \sum_{i=1}^{n-1} 4K_i r^{n+i+3} I_2(n+i-1) \\
&\qquad + \sum_{i=1}^{n-1} K_i r^{n+i+4} I_3(n+i-1) \\
&\qquad + 4K_n r^{2n+2} I_1(2n-1) \\
&\qquad - 4K_n r^{2n+3} I_2(2n-1) \\
&\qquad + K_n r^{2n+4} I_3(2n-1) \\
&\qquad + 4nC_n r^{n+2} I_1(n-1) \\
&\qquad - 4nC_n r^{n+3} I_2(n-1) \\
&\qquad + nC_n r^{n+4} I_3(n-1) \Big) \\
&= \frac{1}{E_{\mathrm{H}}(r,d)} \Bigg(\sum_{i=1}^{n-1} 4K_i r^{n+i+2} \frac{2}{(n+i)(n+i+2)} \\
&\qquad - \sum_{i=1}^{n-1} 4K_i r^{n+i+3} \frac{3\sqrt{\pi}}{2(n+i+1)(n+i+3)} \frac{\Gamma((n+i)/2)}{\Gamma((n+i-1)/2+1)} \\
&\qquad + \sum_{i=1}^{n-1} K_i r^{n+i+4} \frac{8}{(n+i)(n+i+2)(n+i+4)} \\
&\qquad + 4K_n r^{2n+2} \frac{2}{(2n)(2n+2)} \\
&\qquad - 4K_n r^{2n+3} \frac{3\sqrt{\pi}}{2(2n+1)(2n+3)} \frac{\Gamma((2n)/2)}{\Gamma((2n-1)/2+1)} \\
&\qquad + K_n r^{2n+4} \frac{8}{(2n)(2n+2)(2n+4)} \\
&\qquad + 4nC_n r^{n+2} \frac{2}{(n)(n+2)} \\
&\qquad - 4nC_n r^{n+3} \frac{3\sqrt{\pi}}{2(n+1)(n+3)} \frac{\Gamma(n/2)}{\Gamma((n-1)/2+1)} \\
&\qquad + nC_n r^{n+4} \frac{8}{(n)(n+2)(n+4)} \Bigg)
\end{aligned}
$$

$$
\begin{aligned}
= \frac{1}{E_{\mathrm{H}}(r,d)} \Bigg(& \sum_{i=1}^{n-1} 4(-1)^i \binom{n}{i} \frac{2^{i+1}\pi^{(n-i)/2} r^{n+i+2}}{\Gamma((n-i)/2)} \frac{2}{(n+i)(n+i+2)} \prod_{k=1}^{i} \frac{1}{n+i-2k} \\
& - \sum_{i=1}^{n-1} 4(-1)^i \binom{n}{i} \frac{2^{i+1}\pi^{(n-i)/2} r^{n+i+3}}{\Gamma((n-i)/2)} \frac{3\sqrt{\pi}}{2(n+i+1)(n+i+3)} \\
& \qquad \frac{\Gamma((n+i)/2)}{\Gamma((n+i-1)/2+1)} \prod_{k=1}^{i} \frac{1}{n+i-2k} \\
& + \sum_{i=1}^{n-1} (-1)^i \binom{n}{i} \frac{2^{i+1}\pi^{(n-i)/2} r^{n+i+4}}{\Gamma((n-i)/2)} \frac{8}{(n+i)(n+i+2)(n+i+4)} \\
& \qquad \prod_{k=1}^{i} \frac{1}{n+i-2k} \\
& + 4\frac{(-1)^n r^{2n+2}}{(n-1)!} \frac{2}{(2n)(2n+2)} \\
& - 4\frac{(-1)^n r^{2n+3}}{(n-1)!} \frac{3\sqrt{\pi}}{2(2n+1)(2n+3)} \frac{\Gamma((2n)/2)}{\Gamma((2n-1)/2+1)} \\
& + \frac{(-1)^n r^{2n+4}}{(n-1)!} \frac{8}{(2n)(2n+2)(2n+4)} \\
& + 4\frac{n\pi^{n/2} r^{n+2}}{\Gamma(n/2+1)} \frac{2}{(n)(n+2)} \\
& - 4\frac{n\pi^{n/2} r^{n+3}}{\Gamma(n/2+1)} \frac{3\sqrt{\pi}}{2(n+1)(n+3)} \frac{\Gamma(n/2)}{\Gamma((n-1)/2+1)} \\
& + \frac{n\pi^{n/2} r^{n+4}}{\Gamma(n/2+1)} \frac{8}{(n)(n+2)(n+4)} \Bigg)
\end{aligned}
$$

$$
\begin{aligned}
= \frac{1}{E_{\mathrm{H}}(r,d)} \Bigg(& \sum_{i=1}^{n-1} (-1)^i \binom{n}{i} \frac{2^{i+4} \pi^{(n-i)/2} r^{n+i+2}}{(n+i)(n+i+2)\Gamma((n-i)/2)} \prod_{k=1}^{i} \frac{1}{n+i-2k} \\
& - 3 \sum_{i=1}^{n-1} (-1)^i \binom{n}{i} \frac{2^{i+2} \pi^{(n-i+1)/2} r^{n+i+3}}{(n+i+1)(n+i+3)\Gamma((n-i)/2)} \\
& \qquad \frac{\Gamma((n+i)/2)}{\Gamma((n+i-1)/2+1)} \prod_{k=1}^{i} \frac{1}{n+i-2k} \\
& + \sum_{i=1}^{n-1} (-1)^i \binom{n}{i} \frac{2^{i+4} \pi^{(n-i)/2} r^{n+i+4}}{(n+i)(n+i+2)(n+i+4)\Gamma((n-i)/2)} \prod_{k=1}^{i} \frac{1}{n+i-2k} \\
& + \frac{(-1)^n r^{2n+2}}{(n-1)!} \frac{2}{n(n+1)} \\
& - \frac{(-1)^n r^{2n+3}}{(n-1)!} \frac{6\sqrt{\pi}}{(2n+1)(2n+3)} \frac{\Gamma(n)}{\Gamma((2n-1)/2+1)} \\
& + \frac{(-1)^n r^{2n+4}}{(n-1)!} \frac{1}{n(n+1)(n+2)} \\
& + \frac{n \pi^{n/2} r^{n+2}}{\Gamma(n/2+1)} \frac{8}{n(n+2)} \\
& - \frac{n \pi^{n/2} r^{n+3}}{\Gamma(n/2+1)} \frac{6\sqrt{\pi}}{(n+1)(n+3)} \frac{\Gamma(n/2)}{\Gamma((n-1)/2+1)} \\
& + \frac{n \pi^{n/2} r^{n+4}}{\Gamma(n/2+1)} \frac{8}{n(n+2)(n+4)} \Bigg)
\end{aligned}
$$

$$
\begin{aligned}
= \frac{1}{E_{\mathrm{H}}(r,d)} \Bigg(& \sum_{i=1}^{n-1} (-1)^i \binom{n}{i} \frac{2^{i+4} \pi^{(n-i)/2} r^{n+i+2}}{(n+i)(n+i+2)\Gamma((n-i)/2)} \prod_{k=1}^{i} \frac{1}{n+i-2k} \\
& - 3 \sum_{i=1}^{n-1} (-1)^i \binom{n}{i} \frac{2^{i+2} \pi^{(n-i+1)/2} r^{n+i+3}}{(n+i+1)(n+i+3)\Gamma((n-i)/2)} \\
& \qquad \frac{\Gamma((n+i)/2)}{\Gamma((n+i-1)/2+1)} \prod_{k=1}^{i} \frac{1}{n+i-2k} \\
& + \sum_{i=1}^{n-1} (-1)^i \binom{n}{i} \frac{2^{i+4} \pi^{(n-i)/2} r^{n+i+4}}{(n+i)(n+i+2)(n+i+4)\Gamma((n-i)/2)} \prod_{k=1}^{i} \frac{1}{n+i-2k} \\
& + 2 \frac{(-1)^n r^{2n+2}}{(n+1)!} \\
& - 6 \frac{(-1)^n r^{2n+3} \sqrt{\pi}}{(2n+1)(2n+3)\Gamma((2n-1)/2+1)} \\
& + \frac{(-1)^n r^{2n+4}}{(n+2)!} \\
& + \frac{8n\pi^{n/2} r^{n+2}}{n(n+2)\Gamma(n/2+1)} \\
& - \frac{6n\pi^{(n+1)/2} r^{n+3}}{(n+1)(n+3)n} \frac{2}{\Gamma((n-1)/2+1)} \\
& + \frac{8\pi^{n/2} r^{n+4}}{(n+2)(n+4)\Gamma(n/2+1)} \Bigg)
\end{aligned}
$$

$$
\begin{aligned}
= \frac{1}{E_{\mathrm{H}}(r,d)} \Bigg(& \sum_{i=1}^{n-1} (-1)^i \binom{n}{i} \frac{2^{i+4} \pi^{(n-i)/2} r^{n+i+2}}{(n+i)(n+i+2)\Gamma((n-i)/2)} \prod_{k=1}^{i} \frac{1}{n+i-2k} \\
& - 3 \sum_{i=1}^{n-1} (-1)^i \binom{n}{i} \frac{2^{i+2} \pi^{(n-i+1)/2} r^{n+i+3}}{(n+i+1)(n+i+3)\Gamma((n-i)/2)} \\
& \qquad \frac{\Gamma((n+i)/2)}{\Gamma((n+i-1)/2+1)} \prod_{k=1}^{i} \frac{1}{n+i-2k} \\
& + \sum_{i=1}^{n-1} (-1)^i \binom{n}{i} \frac{2^{i+4} \pi^{(n-i)/2} r^{n+i+4}}{(n+i)(n+i+2)(n+i+4)\Gamma((n-i)/2)} \prod_{k=1}^{i} \frac{1}{n+i-2k} \\
& + 2 \frac{(-1)^n r^{2n+2}}{(n+1)!} \\
& - 6 \frac{(-1)^n r^{2n+3} \sqrt{\pi}}{(2n+1)(2n+3)\Gamma((2n-1)/2+1)} \\
& + \frac{(-1)^n r^{2n+4}}{(n+2)!} \\
& + \frac{8 \pi^{n/2} r^{n+2}}{(n+2)\Gamma(n/2+1)} \\
& - \frac{12 \pi^{(n+1)/2} r^{n+3}}{(n+1)(n+3)\Gamma((n-1)/2+1)} \\
& + \frac{8 \pi^{n/2} r^{n+4}}{(n+2)(n+4)\Gamma(n/2+1)} \Bigg)
\end{aligned}
$$

$$
\begin{aligned}
E[q] = \frac{1}{E_{\mathrm{H}}(r,d)} \Bigg(& \sum_{i=1}^{n-1} (-1)^i \binom{n}{i} \frac{2^{i+4} \pi^{(n-i)/2} r^{n+i+2}}{\Gamma((n-i)/2)} \prod_{k=-1}^{i} \frac{1}{n+i-2k} \\
& - 3 \sum_{i=1}^{n-1} (-1)^i \binom{n}{i} \frac{2^{i+2} \pi^{(n-i+1)/2} r^{n+i+3}}{(n+i+1)(n+i+3)\Gamma((n-i)/2)} \\
& \qquad \frac{\Gamma((n+i)/2)}{\Gamma((n+i-1)/2+1)} \prod_{k=1}^{i} \frac{1}{n+i-2k} \\
& + \sum_{i=1}^{n-1} (-1)^i \binom{n}{i} \frac{2^{i+4} \pi^{(n-i)/2} r^{n+i+4}}{\Gamma((n-i)/2)} \prod_{k=-2}^{i} \frac{1}{n+i-2k} \\
& + 2 \frac{(-1)^n r^{2n+2}}{(n+1)!} \\
& - 6 \frac{(-1)^n r^{2n+3} \sqrt{\pi}}{(2n+1)(2n+3)\Gamma((2n-1)/2+1)} \\
& + \frac{(-1)^n r^{2n+4}}{(n+2)!} \\
& + \frac{8 \pi^{n/2} r^{n+2}}{(n+2)\Gamma(n/2+1)} \\
& - \frac{12 \pi^{(n+1)/2} r^{n+3}}{(n+1)(n+3)\Gamma((n-1)/2+1)} \\
& + \frac{8 \pi^{n/2} r^{n+4}}{(n+2)(n+4)\Gamma(n/2+1)} \Bigg)
\end{aligned}
$$

This yields the final form for $E[q]$ for the $d > 3$ case. However, we can simplify this formula by first noting that

$$
\begin{aligned}
\frac{2^{i+1}}{\prod_{k=0}^{i} n+i-2k} &= \frac{1}{\prod_{k=0}^{i} (n+i)/2-k} \\
&= \frac{1}{\left(\frac{n+i}{2}\right)\left(\frac{n+i}{2}-1\right)\cdots\left(\frac{n+i}{2}-i\right)} \\
&= \frac{\left(\frac{n+i}{2}-i-1\right)\left(\frac{n+i}{2}-i-2\right)\left(\frac{n+i}{2}-i-3\right)\cdots(1)}{\left(\frac{n+i}{2}\right)\left(\frac{n+i}{2}-1\right)\cdots\left(\frac{n+i}{2}-i\right)\left(\frac{n+i}{2}-i-1\right)\cdots(1)} \\
&= \frac{\Gamma\left(\frac{n-i}{2}\right)}{\Gamma\left(\frac{n+i}{2}+1\right)}
\end{aligned}
$$

If we substitute this into the $E[q]$ formula we get,

$$
\begin{aligned}
E[q] = \frac{1}{E_{\mathrm{H}}(r,d)} \Bigg(& \sum_{i=1}^{n-1} (-1)^i \binom{n}{i} \frac{2^{i+4} \pi^{(n-i)/2} r^{n+i+2}}{\Gamma((n-i)/2)} \prod_{k=-1}^{i} \frac{1}{n+i-2k} \\
& - 3 \sum_{i=1}^{n-1} (-1)^i \binom{n}{i} \frac{2^{i+2} \pi^{(n-i+1)/2} r^{n+i+3}}{(n+i+1)(n+i+3)\Gamma((n-i)/2)} \\
& \qquad \frac{\Gamma((n+i)/2)}{\Gamma((n+i-1)/2+1)} \prod_{k=1}^{i} \frac{1}{n+i-2k} \\
& + \sum_{i=1}^{n-1} (-1)^i \binom{n}{i} \frac{2^{i+4} \pi^{(n-i)/2} r^{n+i+4}}{\Gamma((n-i)/2)} \prod_{k=-2}^{i} \frac{1}{n+i-2k} \\
& + 2 \frac{(-1)^n r^{2n+2}}{(n+1)!} \\
& - 6 \frac{(-1)^n r^{2n+3} \sqrt{\pi}}{(2n+1)(2n+3)\Gamma((2n-1)/2+1)} \\
& + \frac{(-1)^n r^{2n+4}}{(n+2)!} \\
& + \frac{8 \pi^{n/2} r^{n+2}}{(n+2)\Gamma(n/2+1)} \\
& - \frac{12 \pi^{(n+1)/2} r^{n+3}}{(n+1)(n+3)\Gamma((n-1)/2+1)} \\
& + \frac{8 \pi^{n/2} r^{n+4}}{(n+2)(n+4)\Gamma(n/2+1)} \Bigg)
\end{aligned}
$$

$$
\begin{aligned}
= \frac{1}{E_{\mathrm{H}}(r,d)} \Bigg(& \sum_{i=1}^{n-1} (-1)^i \binom{n}{i} \frac{2^3 \pi^{(n-i)/2} r^{n+i+2}}{\Gamma((n-i)/2)} \frac{\Gamma\left(\frac{n-i}{2}\right)}{\Gamma\left(\frac{n+i}{2}+1\right)} \prod_{k=-1}^{-1} \frac{1}{n+i-2k} \\
& - 3 \sum_{i=1}^{n-1} (-1)^i \binom{n}{i} \frac{2\pi^{(n-i+1)/2} r^{n+i+3}}{(n+i+1)(n+i+3)\Gamma((n-i)/2)} \frac{\Gamma((n+i)/2)}{\Gamma((n+i+1)/2)} \\
& \qquad \frac{\Gamma\left(\frac{n-i}{2}\right)}{\Gamma\left(\frac{n+i}{2}+1\right)} (n+i) \\
& + \sum_{i=1}^{n-1} (-1)^i \binom{n}{i} \frac{2^3 \pi^{(n-i)/2} r^{n+i+4}}{\Gamma((n-i)/2)} \frac{\Gamma\left(\frac{n-i}{2}\right)}{\Gamma\left(\frac{n+i}{2}+1\right)} \prod_{k=-2}^{-1} \frac{1}{n+i-2k} \\
& + 2 \frac{(-1)^n r^{2n+2}}{(n+1)!} \\
& - 6 \frac{(-1)^n r^{2n+3} \sqrt{\pi}}{(2n+1)(2n+3)\Gamma((2n+1)/2)} \\
& + \frac{(-1)^n r^{2n+4}}{(n+2)!} \\
& + \frac{8\pi^{n/2} r^{n+2}}{(n+2)\Gamma(n/2+1)} \\
& - \frac{12\pi^{(n+1)/2} r^{n+3}}{(n+1)(n+3)\Gamma((n+1)/2)} \\
& + \frac{8\pi^{n/2} r^{n+4}}{(n+2)(n+4)\Gamma(n/2+1)} \Bigg)
\end{aligned}
$$

$$
\begin{aligned}
= \frac{1}{E_{\mathrm{H}}(r,d)} \Bigg(& \sum_{i=1}^{n-1} (-1)^i \binom{n}{i} \frac{8\pi^{(n-i)/2} r^{n+i+2}}{(n+i+2)\Gamma\left(\frac{n+i}{2}+1\right)} \\
& - 3\sum_{i=1}^{n-1} (-1)^i \binom{n}{i} \frac{2\pi^{(n-i+1)/2} r^{n+i+3}(n+i)}{(n+i+1)(n+i+3)\Gamma\left(\frac{n+i}{2}+1\right)} \frac{\Gamma((n+i)/2)}{\Gamma((n+i+1)/2)} \\
& + \sum_{i=1}^{n-1} (-1)^i \binom{n}{i} \frac{8\pi^{(n-i)/2} r^{n+i+4}}{(n+i+4)(n+i+2)\Gamma\left(\frac{n+i}{2}+1\right)} \\
& + 2\frac{(-1)^n r^{2n+2}}{(n+1)!} \\
& - 6\frac{(-1)^n r^{2n+3}\sqrt{\pi}}{2^{\frac{2n+1}{2}} 2^{\frac{2n+3}{2}} \Gamma((2n+1)/2)} \\
& + \frac{(-1)^n r^{2n+4}}{(n+2)!} \\
& + \frac{8\pi^{n/2} r^{n+2}}{(n+2)\Gamma(n/2+1)} \\
& - \frac{12\pi^{(n+1)/2} r^{n+3}}{2^{\frac{n+1}{2}} 2^{\frac{n+3}{2}} \Gamma((n+1)/2)} \\
& + \frac{8\pi^{n/2} r^{n+4}}{(n+2)(n+4)\Gamma(n/2+1)} \Bigg)
\end{aligned}
$$

$$
\begin{aligned}
&= \frac{1}{E_{\mathrm{H}}(r,d)} \left(\sum_{i=1}^{n-1} (-1)^i \binom{n}{i} \frac{8\pi^{(n-i)/2} r^{n+i+2}}{2\Gamma\left(\frac{n+i}{2}+2\right)} \right. \\
&\qquad - 6 \sum_{i=1}^{n-1} (-1)^i \binom{n}{i} \frac{\pi^{(n-i+1)/2} r^{n+i+3} (n+i)}{(n+i+1)(n+i+3)^{\frac{n+i}{2}}} \frac{1}{\Gamma((n+i+1)/2)} \\
&\qquad + \sum_{i=1}^{n-1} (-1)^i \binom{n}{i} \frac{8\pi^{(n-i)/2} r^{n+i+4}}{4\Gamma\left(\frac{n+i}{2}+3\right)} \\
&\qquad + 2\frac{(-1)^n r^{2n+2}}{(n+1)!} \\
&\qquad - \frac{3}{2} \frac{(-1)^n r^{2n+3} \sqrt{\pi}}{\Gamma((2n+5)/2)} \\
&\qquad + \frac{(-1)^n r^{2n+4}}{(n+2)!} \\
&\qquad + \frac{8\pi^{n/2} r^{n+2}}{(n+2)\Gamma(n/2+1)} \\
&\qquad - \frac{3\pi^{(n+1)/2} r^{n+3}}{\Gamma((n+5)/2)} \\
&\qquad \left. + \frac{8\pi^{n/2} r^{n+4}}{(n+2)(n+4)\Gamma(n/2+1)} \right) \\
&= \frac{1}{E_{\mathrm{H}}(r,d)} \left(\sum_{i=1}^{n-1} (-1)^i \binom{n}{i} \frac{4\pi^{(n-i)/2} r^{n+i+2}}{\Gamma\left(\frac{n+i}{2}+2\right)} \right. \\
&\qquad - 12 \sum_{i=1}^{n-1} (-1)^i \binom{n}{i} \frac{\pi^{(n-i+1)/2} r^{n+i+3}}{2^{\frac{n+i+1}{2}} 2^{\frac{n+i+3}{2}} \Gamma((n+i+1)/2)} \\
&\qquad + \sum_{i=1}^{n-1} (-1)^i \binom{n}{i} \frac{2\pi^{(n-i)/2} r^{n+i+4}}{\Gamma\left(\frac{n+i}{2}+3\right)} \\
&\qquad + 2\frac{(-1)^n r^{2n+2}}{(n+1)!} \\
&\qquad - \frac{3}{2} \frac{(-1)^n r^{2n+3} \sqrt{\pi}}{\Gamma((2n+5)/2)} \\
&\qquad + \frac{(-1)^n r^{2n+4}}{(n+2)!} \\
&\qquad + \frac{8\pi^{n/2} r^{n+2}}{2\Gamma(n/2+2)} \\
&\qquad - \frac{3\pi^{(n+1)/2} r^{n+3}}{\Gamma((n+5)/2)} \\
&\qquad \left. + \frac{8\pi^{n/2} r^{n+4}}{4\Gamma(n/2+3)} \right)
\end{aligned}
$$

$$
\begin{aligned}
= \frac{1}{E_{\mathrm{H}}(r,d)} \Bigg(& \sum_{i=1}^{n-1} (-1)^i \binom{n}{i} \frac{4\pi^{(n-i)/2} r^{n+i+2}}{\Gamma\left(\frac{n+i}{2}+2\right)} \\
& - \sum_{i=1}^{n-1} (-1)^i \binom{n}{i} \frac{3\pi^{(n-i+1)/2} r^{n+i+3}}{\Gamma((n+i+5)/2)} \\
& + \sum_{i=1}^{n-1} (-1)^i \binom{n}{i} \frac{2\pi^{(n-i)/2} r^{n+i+4}}{\Gamma\left(\frac{n+i}{2}+3\right)} \\
& + \frac{2(-1)^n r^{2n+2}}{(n+1)!} \\
& - \frac{3(-1)^n r^{2n+3}\sqrt{\pi}}{2\Gamma((2n+5)/2)} \\
& + \frac{(-1)^n r^{2n+4}}{(n+2)!} \\
& + \frac{4\pi^{n/2} r^{n+2}}{\Gamma(n/2+2)} \\
& - \frac{3\pi^{(n+1)/2} r^{n+3}}{\Gamma((n+5)/2)} \\
& + \frac{2\pi^{n/2} r^{n+4}}{\Gamma(n/2+3)} \Bigg)
\end{aligned}
$$

Which gives us our final result for $E[q]$ for a given dimension, d, and kernel radius r. Written out as a function of r and d, this is $E_{\mathrm{Q}}(r,d)$ which is the notation used in Chapter 3.