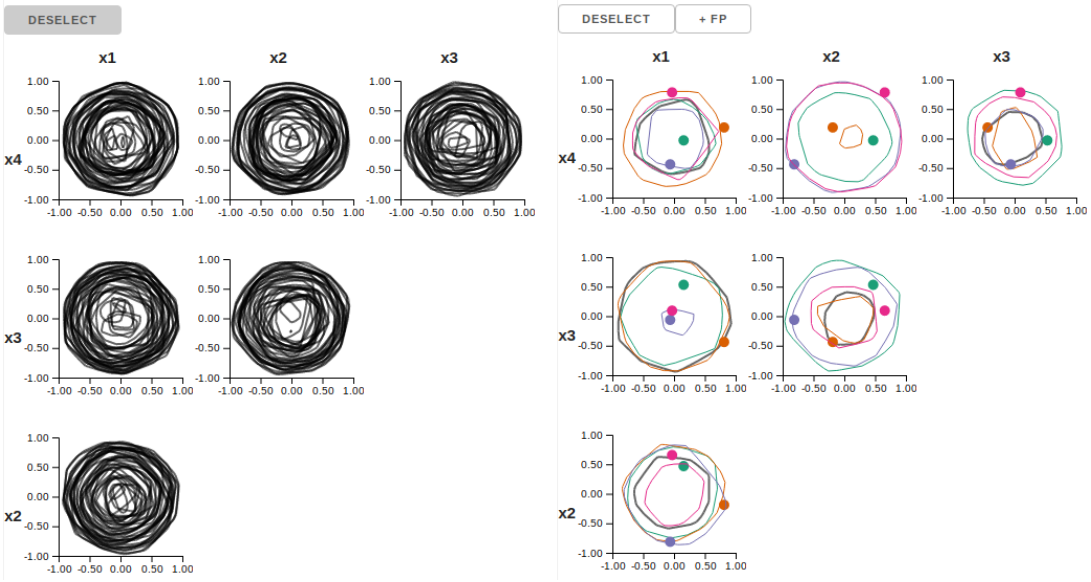
Eurographics Conference on Visualization (EuroVis) 2018
J. Heer, H. Leitte, and T. Ropinski
(Guest Editors)

Volume 37 (2018), Number 3

Hypersliceplorer: Interactive visualization of shapes in multiple dimensions

T. Torsney-Weir[1], T. Möller[1,2], M. Sedlmair[3], and R. M. Kirby[4]

[1]Faculty of Computer Science, University of Vienna, Vienna, Austria
[2]Data Science @ University of Vienna, Vienna, Austria
[3]Computer Science & Electrical Engineering, Jacobs University, Bremen, Germany
[4]School of Computing, University of Utah, Salt Lake City, UT, USA



(a) Global view (b) Local view

Figure 1: The interface for browsing slices created by the Hypersliceplorer algorithm. In this case we show slices of a 4D hypersphere. Each plot shows a pair of dimensions laid out in the same way as HyperSlice [vWvL93]. The interface has two modes: global and local view. The global view (a) shows the results of sampling over a number of focus points. The views are linked through highlighting a slice. The local view (b) shows a single selected slice and then the user can add additional slices by clicking the "+ FP" button. The location of each local focus point is shown as a colored dot and the corresponding slice is also colored accordingly.

Abstract

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

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation

1. Motivation

The visual analysis of multiple dimensions is one of the central themes of visualization research. In principle there are two conceptual types of problems that amount to two different mental models. (1) Often, the data set is considered to be truly discrete and projection methods, such as scatterplots and dimensionality reduction techniques, are used for its analysis. Typical examples include business applications, in which one is analyzing customer data. The focus of this paper is different in that (2) we are focusing on continuous multi-dimensional data spaces. For computational purposes, the data set is then merely a set of points sampled from a continuous phenomenon of study. This is rather common in simulation and engineering applications or for the study of continuous algorithmic parameters in modeling environments, including machine learning applications [SHB*14]. Of course, for such scenarios, projection based visualization might be of help as well. However, they do not respect the mental model of the object of study [TM04].

To comprehend these continuous data spaces, we extend the HyperSlice [vWvL93] method, which presents a number of slices through data space, all connected through one point in data space called the focus point. Slicing has a number of advantages including undistorted views of the space and the preservation of distances. The disadvantage is that only one focus point can be shown at a time. Vastly different views of the object may be seen depending on the location of the focus point. It is difficult to keep track of ones location while navigating multiple dimensions.

In Sliceplorer [TWSM17], Torsney-Weir et al. suggest to present 1D slices instead of 2D slices in such cases. While a 1D slice carries less information than a 2D slice, they could now present a global view of the multidimensional object by over-plotting many 1D slices. This advantage was worth the loss of 2D information. In this paper, we take the idea of more global overviews and revisit 2D slices for closed multi-dimensional objects, whose overall multi-dimensional shape is of great importance. This has been motivated by a number of real-world application scenarios, from comprehension of multi-dimensional polytopes (generalization of polygons to multiple dimensions) by geometers, to applications in computational science, to multi-dimensional Pareto-front analysis. By showing only slices of the outlines of these polytopes we can again create global views of these data sets through over-plotting of many focus points.

We address the issue of selecting a focus point by sampling a number of focus points and producing projections of 2D slices (Figure 1). This *global view* gives the user an overview of the shape of the object without having to navigate manually. Since we are viewing just the outer hull of the object, we can draw these as a projection of a set of 2D slices. We use linked highlighting to show all slices for a particular focus point. In addition, the user can click on a particular slice and switch to the *local view*. The local view begins by showing the particular slice the user clicked on. The user can add additional focus points and select particular slices for comparison. This interaction models Shneiderman's mantra: "overview first, filter and zoom, details on demand" [Shn96].

Our major contribution is an algorithm called Hypersliceplorer for computing the intersection of a 2D slice with a simplical mesh in any dimension (see Section 3 and Figure 2). The issue is that a 2D plane does not have a well-defined normal in spaces higher than three. Therefore, we cannot use the typical point-normal form of a plane in order to compute the intersection of the plane with the simplex. Instead, we treat the plane as a point with two free parameters representing the plane. Then, we show how this representation allows us to compute how a multi-dimensional simplex intersects a 2D plane. This approach lets us compute slices of a multi-dimensional object without a parametric form of the surface. We also demonstrate the results of this algorithm with an interactive interface we developed.

We evaluate our algorithm and interface in two ways representing their recommended evaluation methods according to the nested model [Mun09]. For the overall technique and interactive viewer, we demonstrate the effectiveness of our technique by presenting three case studies. For the underlying algorithm we present an analysis of the running time.

In summary, our contributions are:

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

2. Related work

The study of two- and three-dimensional shapes has been extensively studied by the computer-aided design community [Far96]. Multi-objective optimization and multi-dimensional objects are two areas where it is important to study shapes in over three dimensions. We discuss these areas below. Topological techniques are based on viewing critical points of manifolds [CLB11, GBPW10] or how contours merge and split [CSA03]. We do not discuss them further. Manifold analysis is very different than visualizing shapes.

The need to understand multi-dimensional polytopes is apparent to geometers [Zie12]. However, there are a number of cases in computational science where the understanding of the size and the shape of a sub-section of the parameter space is of importance [BSM*13,SHB*14]. One of these cases is highlighted in Section 5.2. Another use case is the study of multi-dimensional Pareto fronts (Section 5.3).

2.1. Multi-objective optimization

In multi-objective optimization we have several scalar values that we wish to optimize. The set of optimal points is known as the Pareto front. If each objective measure is continuous then we have a continuous hull in one orthant. We want to use this hull to analyze the trade-offs between objective measures. Interactive decision maps [LBK04] show a 3D Pareto front as a series of 2D slices. Any objectives past three must be constrained to a value however. The Prosection matrix took this a step further and shows, for each input configuration, how many of the multiple objectives are satisfied [TS98]. Objective functions are difficult to sample since we often do not have control over the sampling of the range of a function. To generate this hull one often samples the objective functions and

then computes the Pareto points using an algorithm such as NSGA-II [DPAM02] or the skyline algorithm [BKS01]. We can then generate the hull using multi-dimensional marching cubes [BWC00], the quickhull algorithm [BDH96], or alpha shapes [EKS83]. These can then be viewed in Hypersliceplorer as we do in Section 5.3.

An alternative is to treat the samples as a fixed set and then visualize the relationship between possible combinations of objectives. Typically this is done by examining the weight space through interaction. LineUp [GLG*13] uses a ranked list approach and shows the user how rankings will change as the user changes the relative weighting for each objective. WeightLifter [PSTW*16] extends this by also showing the stability of rankings. The user can understand how much a particular objective is affected by its weighting. This can help speed interactive exploration. Finally, the joint contour net [CD14] can be used to compute how often two objectives hold particular values simultaneously. In our case, the mental model is a continuous one. Thus it makes more sense to show a continuous Pareto front.

2.2. Multi-dimensional objects

Three-dimensional polytopes usually result from either the reconstruction of 3D point clouds (see Dey [Dey06]) or from iso-surfacing techniques (see Wenger [Wen13]). There are extensions to iso-surfacing techniques in multiple dimensions [BWC00], but in more than three dimensions we must distort the space somehow to visualize the object.

For the visualization of 4D polytopes, there are a number of techniques for moving from four to three dimensions. The Schlegel diagram [Som29] is one such method based on projection. One picks a face of the figure, usually the largest, which is a three-dimensional object. Then, all other faces are "packed" inside this face in such a way that we can show the connections between faces. The Schlegel diagram works well for regular polytopes where we have some previous intuition about the faces. However, for an arbitrary simplical mesh, any face is a simplex which we need to project into. All Schlegel diagrams of a simplical mesh look like a simplex with a number of other simplices inside them. It can be difficult to recover what the original object looks like because the cross section is lost. An alternative approach is to treat the fourth dimension as time and then produce an animation of the evolution of the shape in three dimensions. In this case each frame of the animation is a 3D slice of the object. Rather than first projecting from 4D to 3D and then rendering the projection, Hanson and Cross [HC93] propose a method to first render the object in 4D and then view the three-dimensional projection. This allows them to show unique lighting effects from the 4D surfaces. As with all projection methods, if the user is unaware of the details of the method it can be difficult to build a mental model of the shape under study.

Hasse diagrams [BT88] are based on showing the connectivity between vertices of an object. These can be seen as network diagrams where the vertices of the figure are the nodes in the graph and the edges of the graph are the edges in the figure. These have a number of layout issues. For visual understanding, humans prefer a 2D planar graph [KDMW16]. Good layouts of the Hasse diagram must balance human aesthetic needs like few edge crossings with the geometric interpretation. There are automatic layout algorithms, such as the one by Battista et al. [BT88], but these do not work in all cases.

For more than four dimensions, projection methods no longer work as well. On the other hand, techniques based on slicing the space can be extended to any number of dimensions. The techniques to perform this so far, such as HyperSlice [vWvL93], HyperMoVal [PBK10], and Sliceplorer [TWSM17], require the multi-dimensional object to be specified as a manifold (i.e. a function). Then, for a given focus point, we can constrain all but one or two parameters. This gives us a one- or two-dimensional function which we can draw as a function plot. The function plot can be drawn using a heatmap, contour plot, or line function plot. In our case we have a simplical mesh for which we cannot compute the slice by constraining all but a few parameters to a value. We need a new way to compute the slice of a simplex in multiple dimensions which cannot be done by constraining parameters. Further, HyperSlice can only show the slices around a single focus point at a time. Hypersliceplorer can show slices for multiple focus points at once.

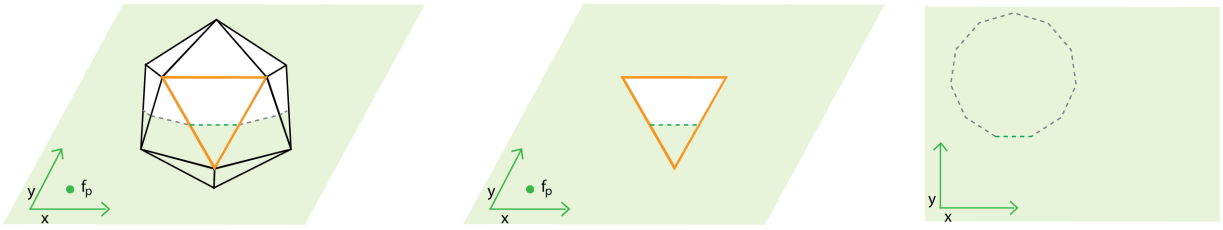
Sliceplorer addressed the focus point issue by sampling over a number of focus points and projecting them down. Exploded view diagrams [KLMA10] offer a hybrid method between a 3D volume visualization and slicing. However, they are limited to 3D objects. The global view of Hypersliceplorer is inspired by the idea of examining cross sections. We also have a local view which permits the user to look at a small number of self-selected slices. We have developed a method to produce slices based on a simplical mesh which is very useful given a discretized surface (see Figure 2).

3. Algorithm

Our main contribution is an algorithm that computes 2D axis-aligned slices of a multi-dimensional polytope in the form of a simplical hull. This hull can be generated from a point cloud or it can be pre-computed. In any case, a simplical hull is a set of $(d-1)$-dimensional simplices in a d-dimensional space. We compute the intersection of a two-dimensional plane with this set of $(d-1)$-dimensional simplices (see Figure 2a). The algorithm produces a single slice for a single pair of dimensions and focus point selection. To produce the multi-dimensional view we repeat this algorithm for each focus point and dimension pair.

One way to compute the intersection of a plane with a simplex is to check, for each pair of points, if they are on opposite sides of the plane using the point-normal form of the plane. However, in more than three dimensions a 2D plane does not have a well-defined concept of a "side." The analogy to this is the normal of a line in three-dimensions. Therefore, we cannot use the point-normal form to define the 2D axis-aligned plane that represents the slice.

Our solution to this problem relies on two key observations. We can represent the axis-aligned plane as a point with two free parameters and then see if this point lies on the boundary of the simplex using barycentric coordinates. If the plane intersects the simplex then it must intersect the simplex at its boundaries. Therefore, it is sufficient to set each barycentric coordinate to 0 in turn to compute where, if any, is the line of intersection between the plane and sim-



(a) Slicing a mesh (b) Slicing a simplex (c) 2D slice

Figure 2: An overview of how our algorithm functions. The goal is to compute the intersection of a slice with a polytope defined as a simplical mesh (a). The slice is defined by selecting a focus point and then extending it in two directions. We (b) treat each simplex in the mesh independently and compute the intersection of the simplex with the slice (see Algorithm 2). The collection of all intersections for a particular plane is shown as a line plot (c). This process is repeated over a number of randomly sampled focus points.

plex. We will first explain the mathematical basis before providing implementation details.

3.1. Mathematical basis

The basis of our mathematical derivations can be best described by intersecting a 2D axis-aligned plane with a d-dimensional simplex in a d-dimensional space. The intersection of a d-dimensional simplex with a 2D plane will result in a 2D object [Han94]. An intersection with a plane must pass through a boundary face of the simplex. The implementation section will discuss how to extend a $(d-1)$-dimensional simplex to a d-dimensional simplex (Section 3.2).

To illustrate the mathematics behind our slicing algorithm we first introduce some notation. We begin with a d-dimensional focus point f_p and a simplex s consisting of $d+1$ d-dimensional points, $x_1,\ldots,x_{d+1}$. We denote the slice coordinates as f'_p in the formulae below. Without loss of generality, we will assume the slicing dimensions to be $(d_1,d_2)=(1,2)$. Then, the two free variables for the specification of the slice, x and y, will replace the first two components of the focus point (Equation 2). We let T be the matrix to convert a point from barycentric coordinates to Cartesian coordinates (including a homogeneous component of 1). The columns of T are the $d+1$ points defining the simplex. We append a row of ones to ensure that the barycentric coordinates sum to one. If T is nonsingular then the simplex is not degenerate and we can easily invert it. The inverse of T will convert a point from Cartesian coordinates (including a homogeneous component of 1) back to barycentric coordinates (t is denoting the transpose).

$$f_p = [p_1, p_2, \ldots, p_d, 1]^t \tag{1}$$

$$f'_p = [x, y, p_3, \ldots, p_d, 1]^t \tag{2}$$

$$T = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,d+1} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,d+1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{d,1} & x_{d,2} & \cdots & x_{d,d+1} \\ 1 & 1 & \cdots & 1 \end{bmatrix} \tag{3}$$

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

$$T^{-1} = \begin{bmatrix} \alpha_{1,1} & \alpha_{1,2} & \cdots & \alpha_{1,d+1} \\ \alpha_{2,1} & \alpha_{2,2} & \cdots & \alpha_{2,d+1} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{d+1,1} & \alpha_{d+1,2} & \cdots & \alpha_{d+1,d+1} \end{bmatrix} \tag{4}$$

$$\lambda = T^{-1} f'_p \tag{5}$$

$$= \begin{bmatrix} \alpha_{1,1}x & +\alpha_{1,2}y & +\alpha_{1,3}p_3 & +\cdots & +\alpha_{1,d+1} \\ \alpha_{2,1}x & +\alpha_{2,2}y & +\alpha_{2,3}p_3 & +\cdots & +\alpha_{2,d+1} \\ \vdots & & & & \\ \alpha_{d+1,1}x & +\alpha_{d+1,2}y & +\alpha_{d+1,3}p_3 & +\cdots & +\alpha_{d+1,d+1} \end{bmatrix} \tag{6}$$

This expression for λ is linear in x and y and we denote its coefficients by a_x, a_y, and a_c respectively. Thus, each component expresses a line.

$$a_x = \left[\alpha_{1,1}, \alpha_{2,1}, \ldots, \alpha_{d+1,1}\right]^t \tag{7}$$

$$a_y = \left[\alpha_{1,2}, \alpha_{2,2}, \ldots, \alpha_{d+1,2}\right]^t \tag{8}$$

$$a_c = \left[\sum_{i=3}^{d+1} \alpha_{1,i}p_i, \ldots, \sum_{i=3}^{d+1} \alpha_{d+1,i}p_i\right]^t \tag{9}$$

$$\lambda = a_x x + a_y y + a_c \tag{10}$$

where we assume $p_{d+1}=1$. Further, x and y correspond to the horizontal and vertical coordinates of the intersection of the plane with the simplex. Each component of λ reflects the influence of one of the $(d+1)$ points of the simplex. If the influence is zero (i.e. for the i-th point we have $\lambda_i=0$) then we are on the boundary of the simplex. An intersection of a plane with a simplex must cross the boundaries (Figure 2b). It is possible that the plane will intersect multiple faces so we set each component λ_i of λ to 0 in turn and then check what pairs of x and y in the remaining components are valid barycentric coordinates (i.e. are between zero and one). If this is the case, then the plane intersects the simplex. Otherwise, there is no intersection. In other words, for face i, we need to solve $\lambda_i=0$ such that $\forall j \neq i, 0 \leq \lambda_j \leq 1$.

Setting a particular λ_i to 0 and solving for y yields:

$$\lambda_i = a_{x_i}x + a_{y_i}y + a_{c_i} = 0 \tag{11}$$

$$y = -\frac{a_{x_i}x + a_{c_i}}{a_{y_i}}. \tag{12}$$

Then, we substitute this for y in the remaining $\lambda_j, j \neq i$,

$$\lambda_j = a_{x_j}x + a_{y_j}y + a_{c_j} \tag{13}$$

$$= a_{x_j}x + a_{y_j}\left(-\frac{a_{x_i}x + a_{c_i}}{a_{y_i}}\right) + a_{c_j} \tag{14}$$

$$\lambda_j = \left(a_{x_j} - \frac{a_{y_j}a_{x_i}}{a_{y_i}}\right)x + \left(a_{c_j} - \frac{a_{y_j}a_{c_i}}{a_{y_i}}\right) \tag{15}$$

$$= b_{x_j}x + b_{c_j}, \tag{16}$$

where b_{x_j} and b_{c_j} denote the coefficients of the reduced equation. In order to be on the face, all λ_j must be between 0 and 1. Assuming $b_{x_j} > 0$, we now find the range for each $0 \leq \lambda_j \leq 1$,

$$0 \leq \lambda_j \leq 1 \tag{17}$$

$$0 \leq b_{x_j}x + b_{c_j} \leq 1 \tag{18}$$

$$-\frac{b_{c_j}}{b_{x_j}} \leq x \leq \frac{1 - b_{c_j}}{b_{x_j}}. \tag{19}$$

If b_{x_j} is negative, then the inequalities are reversed. For each λ_j we will get a range of valid x. The intersection of these ranges over all j gives the valid range for x. If this range is non-empty then we can substitute this range of x into Equation 12 to find the corresponding range for y and hence a line of the intersection of the slice with the d-dimensional simplex.

3.2. Algorithm details

Our implementation is based on intersecting a 2D plane with a $(d-1)$-dimensional simplex as opposed to a d-dimensional simplex. In that case, our matrix in Equation 3 is a non-square matrix and can not easily be inverted. Hence, we add an additional point, the focus point, to create a d-dimensional simplex. Before we draw any slices, however, we remove all those intersections that include the sides created by the extra point.

The outer loop of the algorithm, shown in Algorithm 1, loops over each combination of simplex, pair of slicing dimensions, and focus points. We select these focus points by sampling. When sampling in a multi-dimensional space it is important to be economical with sample points. These methods are based on ensuring that the distance between sample points is as even as possible [SWN03]. There are several ready-made solutions to creating a number of uniformly distributed multi-dimensional samples. In our implementation we used the Sobol sequence but other methods such as Latin hypercube sampling [MBC79] are available. Each step of the loop is independent and thus our algorithm can be easily parallelized.

We show the algorithm for the 2D slice/simplex intersection in Algorithm 2. This algorithm implements the mathematics from Section 3.1. The function `SOLVE` checks for the plane intersection using a linear constraint solver. In the case that there is no intersection, it will return a null range. Otherwise, it will return a pair of points representing the line segment. The `POINT-IN-SEG` function checks if the given point intersects the given line segment. We can then draw these intervals as line segments on the screen (Figure 2c).

Algorithm 1 Finding slices for all simplices.

```
for d1 = 1 to d − 1 do
    for d2 = d1 to d do                          ▷ all pairs of dimensions
        slices ← [∅, ∅, ∅, ∅]   ▷ 4 element array for min/max x and y
        for p ∈ FP do                            ▷ all focus points
            for s ∈ S do                         ▷ all simplices
                ranges ← SLICE(p, s, d1, d2)
                if ranges ≠ ∅ then ▷ add new row if we found an intersection
                    slices ← ADD_ROW(slices, ranges)
                end if
            end for
        end for
        PLOT(slices, d1, d2)                     ▷ plot slices to proper subplot
    end for
end for
```

Algorithm 2 Slicing a single simplex

```
function SLICE(p, s, d1, d2)
    T ← [s p 1]^t                                ▷ add focus point to simplex
    if IS_SINGLULAR(T) then
        return GET_COLUMNS(T, d1, d2) ▷ the simplex lies on the plane
    end if
    p' ← p
    p'[d1, d2] ← [x, y]
    a_x x + a_y y + a_c ← T^{-1} p'              ▷ convert to barycentric coordinates
    lb ← [−∞, ∞]                 ▷ create 2 points for the intersection
    ub ← [−∞, ∞]
    for i ← 1 to d + 1 do                        ▷ each face of the simplex
        (lb', ub') ← SOLVE(a_{x_i} x + a_{y_i} y + a_{c_i} = 0, s.t. ∀ j ≠ i, 0 ≤ a_{x_j} x + a_{y_j} y + a_{c_j} ≤ 1)
        if lb' ≠ ∅ ∧ ub' ≠ ∅ ∧ ¬POINT-IN-SEG(p, (lb', ub')) then
            (lb, ub) ← (lb, ub) ∩ (lb', ub')     ▷ 2D range intersection
        end if
    end for
    if lb = ∅ ∨ ub = ∅ then
        return ∅
    else
        return (lb, ub)
    end if
end function
```

4. Interface

We developed an interactive viewer to browse and select slices of interest in order to build up an understanding of the object we are viewing. Slicing is an inherently interactive operation. Depending

on the focus point we will see different aspects of the data. In a multi-dimensional space, it is easy to get lost navigating freely without guidance. However, if we show all slices at once the user cannot closely examine one particular aspect of the data. Thus, our interactive interface, shown in Figure 1, has two modes: a global view and a local view. The global view is designed to give an overview of the general shape. By selecting a slice and corresponding focus point of interest, the user can then switch to the local view and gradually add additional slices at new focus points.

4.1. Global view

The global view (Figure 1a) gives an overview of the possible cross sections of the object. By default we show slices for the first 50 focus points sampled using a Sobol sequence [Sob67]. We experimentally found that 50 focus points was sufficient for an overview. This has the advantage of generating equally spaced samples as well as the incremental addition of sample points does not require previous samples to be thrown away. Since we are slicing hulls of simplical meshes, each slice is a contiguous line plot in the view. We use alpha blending in order to show the distribution of hull shapes in each pair of dimensions. From this the user can get insight into whether or not a shape has a regular structure.

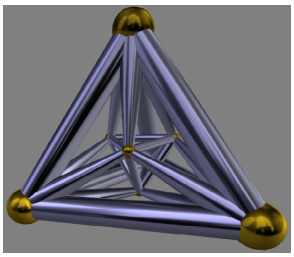
With more than one slice one cannot easily tell how the slices correspond between panels in the layout. We address this by using linked highlighting between the plots. If the user mouses over a slice in one plot the slices corresponding to that focus point are highlighted in the other plots. In addition,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

4.2. Local view

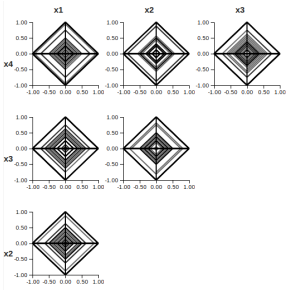
The local view mode of the interface allows the user to narrow in on a particular focus point and then explore how other slices of the figure relate to that one. We precompute the slices so that the view can be interactive. The focus point is represented as a dot projected on each sub plot. The user can change the focus point by dragging the focus point dot to a new location. After the user releases the mouse the system will snap to the nearest precomputed focus point. Thus, the user can change one or two focus point values per dragging interaction. The user can also add additional focus points by clicking the "+ FP" button in the upper left of the interface. Each focus point is automatically colored based on a discrete color map from ColorBrewer [HB03]. The slices themselves and the focus points are linked through a similar color. For example, one mode of exploration this view supports is examining the faces orthogonal to one of the slices. The user can return to the global view by clicking the "deselect" button on the top left of the interface.

5. Case studies

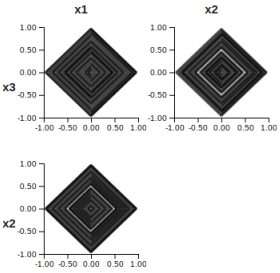
Three areas where our method can be used is in the visual analysis of multi-dimensional polytopes, differences between spaces, and multi-objective optimization. We examine each of these in turn in the sections below.



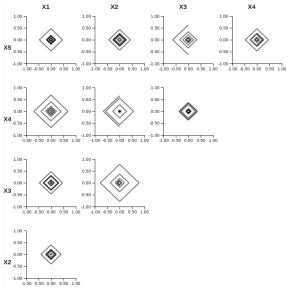
(a) Schlegel diagram: 16-cell (4D) generated using Stella4D [Web17]



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

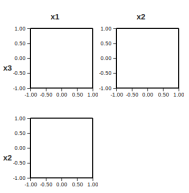


(c) Hypersliceplorer: Octahedron (3D)

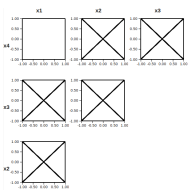


(d) Hypersliceplorer: 5-orthoplex (5D)

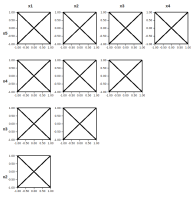
Figure 3: We show the 16-cell, the 4-dimensional regular orthoplex (4D version of an octahedron) as (a) a Schlegel diagram and (b) the Hypersliceplorer view. The Hypersliceplorer view shows the outside shape of the figure and the repeating structure. We can also see the repeating structure in the 3D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



(a) 3D

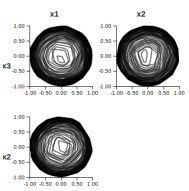


(b) 4D

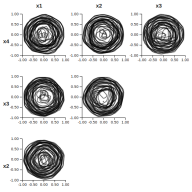


(c) 5D

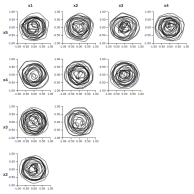
Figure 4: 3-, 4-, and 5-dimensional hypercubes. We can see the regular structure in the cubes. The cross sections are all the same size since the cube is oriented to the axes. The cross lines in the plots are due to the simplical mesh.



(a) 3D



(b) 4D



(c) 5D

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

5.1. Polytopes

Polytopes are the generalization of polygons and polyhedra to any number of dimensions. Two key questions where visualization can help are understanding under what transformations is the polytope invariant and how many faces does it have [Zie01]? Naturally, in more than three dimensions we have no way of viewing these objects directly.

Common ways to view polytopes are either through projections, such as the Schlegel diagram [Som29], or as a graph representation, like the Hasse diagram [BT88]. Projection methods show a particular face of the polytope in detail. For example, the Schlegel diagram picks a particular face of a 4D polytope and projects the remaining faces inside it. On the other hand, it does not accurately show distances or angles. Thus, these don't necessarily show the symmetries in the polytope. These must be distorted to show a multi-dimensional object in two or three dimensions. Network diagrams allow us to count the faces of a polytope by counting the links in the graph. They can also show symmetries in the geometry. However, they do not necessarily show these symmetries unless care is taken during layout. The layout issue makes these difficult to use for exploratory analysis since one must first understand the symmetry and structure of the object before visualizing it. Regular polytopes have a well-studied structure and symmetries so we use these as verification examples.

We can also view the polytopes as a set of two-dimensional slices in Hypersliceplorer. The global view gives an overview of the possible 2D slices of the polytope from which we can better understand what types of symmetries are possible. For example, in Figure 3 we show a 16-cell which is the four-dimensional version of an octahedron in both Hypersliceplorer (Figure 3b) and as a Schlegel diagram (Figure 3a) using the Stella4D software [Web17]. In this case, each face of the 16-cell is a simplex. From the Schlegel diagram it is difficult to see that the 16-cell has rotational symmetry every 90 degrees. However, in Hypersliceplorer this property is clear from looking at the cross sections. In addition, we can see the simplical faces from the horizontal and vertical lines in the view. These result from intersections of the 2D slice with a face of the 16-cell directly. We can use the local view of Hypersliceplorer to focus on a particular face and examine the slices intersecting that face.

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

We can also look at other regular polytopes in the same fashion. In Figure 4 we show a hypercube in 3-, 4-, and 5-dimensions. From these plots we can clearly see the generalization of the square, to the cube, to higher dimensions. One of the advantages of our method is that we do not need to choose a face to project into. For example, with a discretized hypersphere, there are many faces. We can see in Figure 5 the regular cross sections of a sphere as well. The Hypersliceplorer visualization is better suited for exploring and identifying symmetry groups while the Hasse and Schlegel diagrams are better for quickly identifying the number of faces.

5.2. Positive and Bernstein polynomials

In physics, it is sometimes necessary to fit data with a function that is positive everywhere on its domain. An example of such data is density; it is a positive quantity and any regression on it should be positive. In numerical methods, we often use polynomials as a means of representation, and hence we want to find polynomials that are positive on some compact domain, without loss of generality say, $[0,1]$. It is very difficult to control a polynomial such that it is positive during the fitting process using a linear constraint solver. One method used by physicists is to restrict the fitting process to Bernstein polynomials [Phi03]. By only using positive Bernstein coefficients, Bernstein polynomials are restricted to be strictly positive. Bernstein polynomials have been studied extensively for over 50 years. It is well-known how these polynomials can be used to approximate continuous functions, for example, in the creation of Bezier curves [Far96]. However, physicists do not know how "representative" are the positive Bernstein expansions of the space of positive polynomials. In other words, can every positive polynomial be represented by a corresponding Bernstein polynomial? We can select a large number of polynomials and visually compare the spaces in order to understand the differences.

A Bernstein polynomial of degree n is a linear combination of $n+1$ basis polynomials.

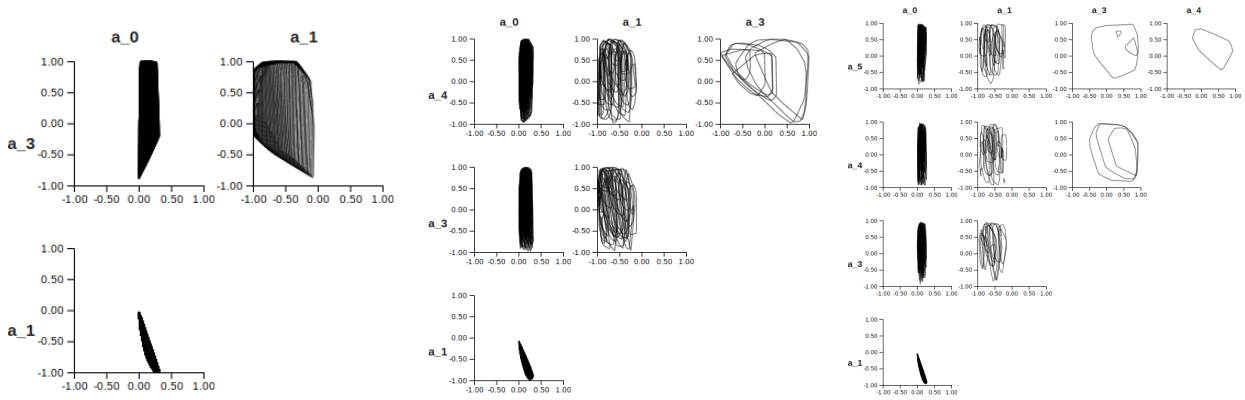
$$B_n(x) = \sum_{i=0}^{n} \beta_i b_{i,n}(x)$$

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

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

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

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



(a) degree 3 polynomials (b) degree 4 polynomials (c) degree 5 polynomials

Figure 6: Examining differences in the space of general positive polynomials and Bernstein polynomials with positive Bernstein coefficients. In this example the x^2 term is set to 1. We can see that across degrees of polynomials, the space differences in the a_0 and a_1 coefficients is relatively consistent. The single slice in c for the a_3, a_4 plot is because the focus point sampling rarely hit a particular slice. The solution is to add additional focus point samples.

the space of polynomials. These faces are convex so we generate a convex hull of these points and examine them using Hypersliceplorer. We are interested in seeing the differences and thus it makes sense to show the difference views explicitly [GAW*11]. The differnce view shows the convex hull around the set of all coefficients that produce a positive polynomial but cannot be represented as a Bernstein polynomial with positive coefficients.

While there is a difference between the set of all positive polynomials and the set of all Bernstein polynomials with positive coefficients, currently, that difference is assumed to be small. However, our Hypersliceplorer visualization shows that this is not the case. Figure 7a shows all 4^{th} degree polynomials with the x^4 term fixed to 1. We can see that for almost an entire range of the x^2 and x^3 coefficients a_1, a_2 there are positive polynomials for which we cannot find a Bernstein polynomial with positive Bernstein coefficients. Using the local view (Figure 7b), we can see that this difference is also large in the other dimensions. Further, other patterns become apparent, such that the space of positive, non-Bernstein polynomials requires $a_0 > 0$ which could lead to novel hypotheses that can be tested.

The global overview also lets us compare across degrees of polynomials. In Figure 6 we show a 3^{rd}, 4^{th}, and 5^{th} degree polynomial with the coefficient of the x^2 term set to 1. Here we can see that the $a_0 \times a_1$ plots all look the same. In fact, the width across all the panels including a_0 are the same. In this case this means that for these ranges of a_0 (the constant term) we will not be able to find a Bernstein polynomial with positive Bernstein coefficients no matter the degree of polynomial.

This information is novel to the domain scientists with whom we interact. They had previously run a number of numerical experiments to try and understand the volume and shape characteristics of the difference space. However, once they saw the visualizations of the polytope they realized that they needed a different method than using Bernstein polynomials with positive Bernstein coefficients.

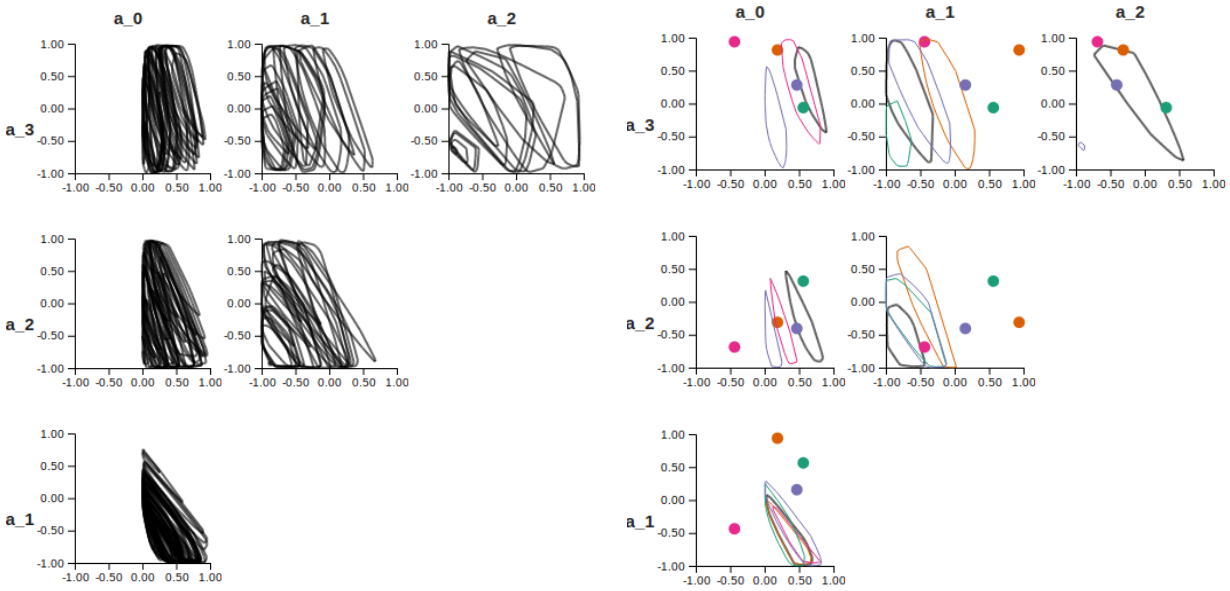
5.3. Pareto fronts

A Pareto front (also known as the efficient frontier) is the set of all points that are optimal with respect to some trade-off between objectives. Algorithms such as the skyline algorithm [BKS01] or NSGA-II [DPAM02] can extract these points automatically. It is often difficult to obtain insight into the trade-offs among multiple objectives when searching for an optimum. Thus visual analysis of the trade-offs is necessary. With two objectives, this can be visualized using a scatterplot or line plot of the two objectives against each other.

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

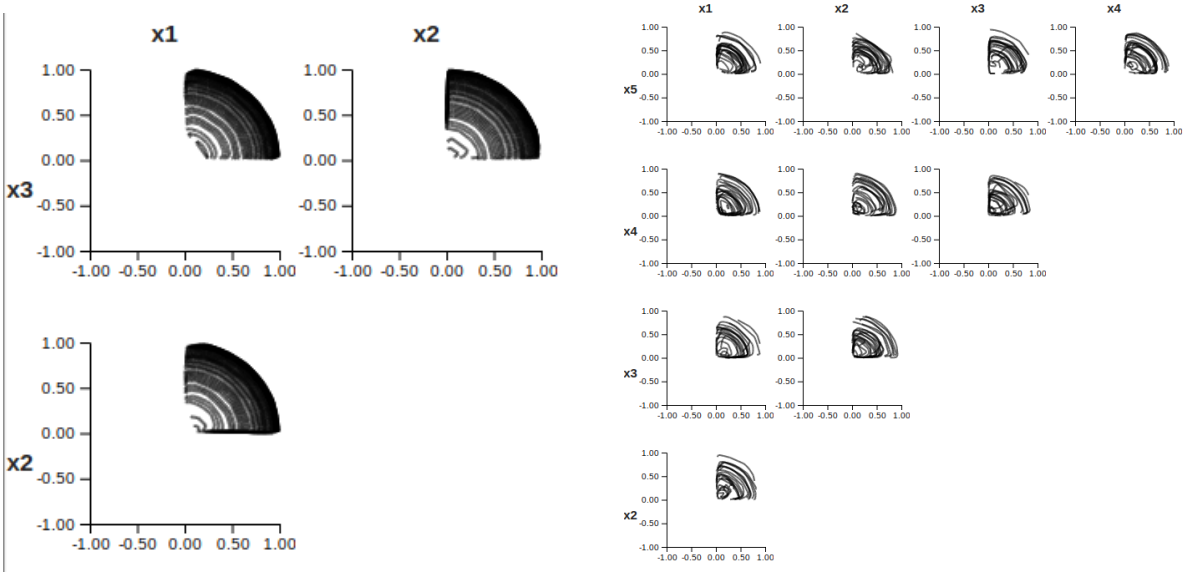
The smoothest possible Pareto front is a sphere in the positive orthant of the objective space. In order to illustrate our technique we show a 3D and 5D positive orthant section of a sphere in Figure 8. Each arc is the trade-off holding all other parameters fixed. In the multi-dimensional view, setting the focus point is analogous to fixing all but two parameters. Thus, in one plot one can directly see the trade-off between two of the objective measures given that all other objectives are held in place. However, the user can also see at a glance what are the *possible* trade-off curves for a pair of objectives. This helps the user to understand what are the costs and benefits of changing one of the remaining objectives.

We also show a popular multi-objective test problem, DLTZ1 [DTLZ02] with 5 objectives. We find the Pareto points using the NSGA-II [DPAM02] algorithm. We used the same settings for the algorithm as in Deb et al. [DTLZ02]. In real-world situations, the Pareto front is not convex. We can use, for example, alpha shapes [EKS83] to generate a non-convex hull of a set of points. For this example, we use the convex hull of the points generated using the quickhull [BDH96] algorithm since the Pareto



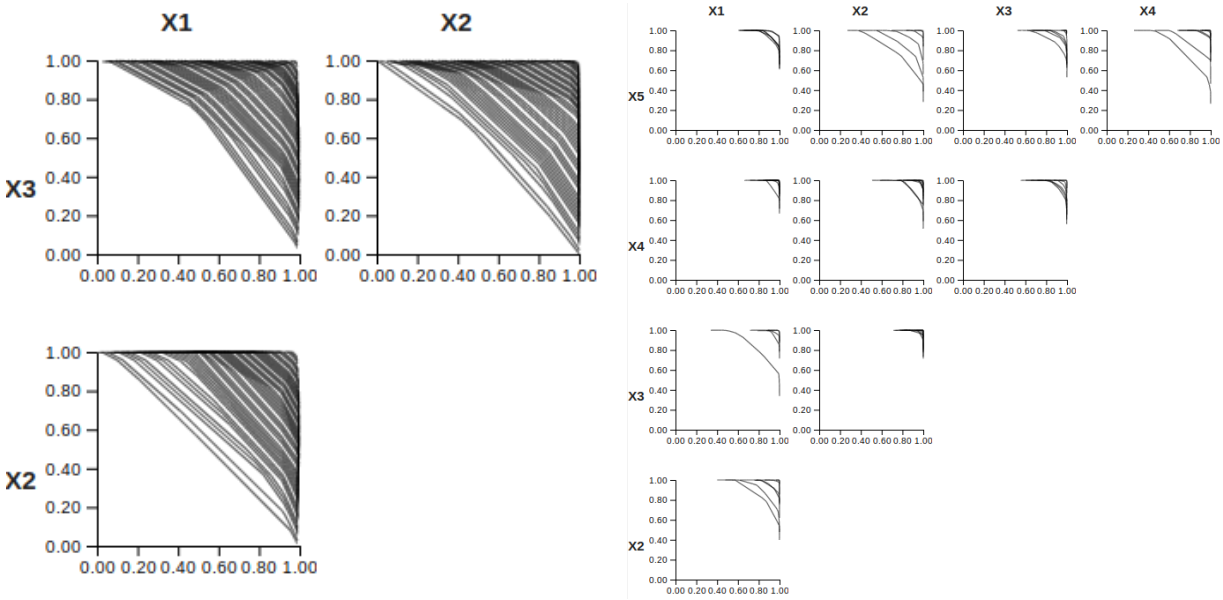
(a) global view (b) local view

Figure 7: The difference between possible coefficient values for general positive polynomials, $a_0 + a_1x + a_2x^2 + a_3x^3 + x^4$, and polynomials that can be represented with positive Bernstein coefficients. From the global view (a) we can see that the area of the slices is quite large. This means that the difference between spaces is quite large, especially with respect to the higher-order coefficients. We can focus our view on a few particular slices in the local view (b) where we can see the orthogonal faces to the slice in the $a_2 \times a_3$ plot.



(a) 3D (b) 5D

Figure 8: Hypersliceplorer views of spherical Pareto fronts in 3D (a) and 5D (b). The smoothest possible Pareto front is the positive orthant of a hypersphere. From the Hypersliceplorer view we can clearly see the concentric arcs. Each arc allows the user to compare the trade-offs between two objectives given that all other objectives are fixed. Changing from one arc to another means changing other objective settings.



(a) 3-objective (b) 5-objective

Figure 9: Visualization of the 3-objective (a) and 5-objective (b) DLTZ1 problems [DTLZ02] in Hypersliceplorer. The DLTZ1 Pareto front is a hyperplane that cuts diagonally across the objective dimensions. In the 3-objective case we can see the slices of the hyperplane. The NSGA-II [DPAM02] algorithm tends to push points towards one objective. This becomes much more pronounced in higher dimensions (b) where the Pareto front is squared off.

front of DLTZ1 is convex. The Pareto front is a hyperplane that cuts diagonally across the objective dimensions. In the 3-objective (Figure 9a) we can see how the NSGA-II algorithm is getting close to fitting the points to the hyperplane. It does not find it exactly though which is why the lines appear bent in the figure. In addition, NSGA-II pushes some points out to the maximum value for each objective. This creates the horizontal and vertical lines at the edges of the plots. This becomes much more pronounced in higher dimensions (Figure 9b) where the Pareto front appears as a corner shape.

6. Algorithm performance

In order to test the running time of our algorithm we ran a number of experiments to understand the timing. We tested regular polytopes in 3, 4, and 5 dimensions as well as hyperspheres. We also tested the four-dimensional version of the Klein bottle because it has a large number of simplices in its mesh. For each test, we ran the slicing algorithm for all pairs of dimensions and for $1,000$ focus points. We recorded the total wall clock time as well as the number of simplices given by the quickhull algorithm. The testing machine has an 8-core 3.2GHz Intel i7-6900K with 64GB of RAM.

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

As we can see, the times are roughly constant between the number of dimensions and simplices (see Figure 10). The reason the Klein bottle is faster is because many slices do not hit any simplices and the algorithm will exit early once this is detected. Right now, this algorithm is not optimized. For example, it would be greatly beneficial to pre-compute a spatial data structure so that only slices

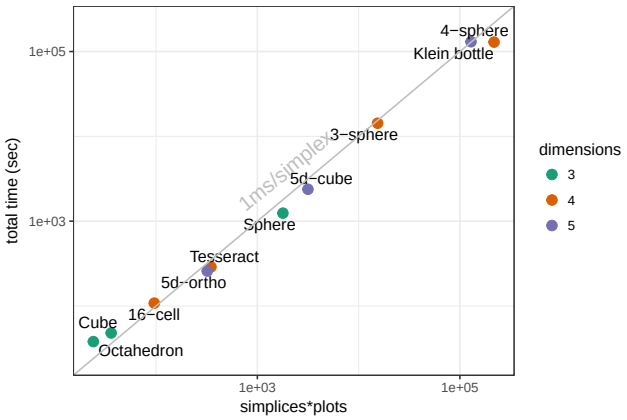


Figure 10: Chart showing the number of slicing operations (simplices × number of plots) versus timing results from running our algorithm on a number of different datasets. The axes are on a log-log scale. The points are all clustered around the "1 ms/simplex" line showing that the running time is roughly one millisecond per slicing operation.

Table 1: Results of timing the Hypersliceplorer algorithm on a number of different datasets. We ran the algorithm, setting the number of slices to 1,000. We record the number of simplices created by the quickhull algorithm [BDH96], the total time for all slices, and the time per simplex. The time per simplex is the total time divided by the number of plots (i.e. dimension pairs, the number of simplices, and the number of focus points. The time per simplex is roughly constant.

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

that are likely to intersect simplices are evaluated. In our case, the algorithm must check every simplex against every focus point for every pair of dimensions. This is a lot of extra work for figures such as the Klein bottle with many simplexes.

7. Conclusion and future work

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

In the future we will improve the speed of our algorithm by integrating a spatial data structure such as a k-d tree or bounding box method. The current algorithm needs to try and find an intersection with every simplex in the figure. The data structure will help to avoid these extra checks. This should improve the speed of our algorithm by avoiding unnecessary intersection tests. Our aim is to make the algorithm run in interactive time.

Currently, we have only tested our method with convex hulls of shapes. There is nothing limiting our algorithm to non-convex hulls. As long as the hull of the multi-dimensional object can be defined as a set of simplices then our algorithm can compute the slice. We plan to also examine the visualization possibilities with non-convex hulls and with pre-generated hulls. Perhaps our method will lead to new insights in mesh generation algorithms.

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

Acknowledgements

We wish to thank the members of the VDA lab for their helpful feedback and support with this project. The authors wish to especially thank Johanna Schlereth for her help in producing some of the diagrams. R. M. Kirby was supported in part by DMS-1521748 and W911NF-15-1-0222.

References

[BDH96] BARBER C. B., DOBKIN D. P., HUHDANPAA H.: The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software 22*, 4 (Dec. 1996), 469–483. doi:10.1145/235815.235821. 3, 8, 11

[BKS01] BORZSONY S., KOSSMANN D., STOCKER K.: The skyline operator. In *Proceedings of the 17th International Conference on Data Engineering* (2001), IEEE Computuer Society. doi:10.1109/icde.2001.914855. 3, 8

[BSM*13] BERGNER S., SEDLMAIR M., MÖLLER T., ABDOLYOUSEFI S. N., SAAD A.: ParaGlide: Interactive parameter space partitioning for computer simulations. *IEEE Transactions on Visualization and Computer Graphics 19*, 9 (Sept. 2013), 1499–1512. doi:10.1109/tvcg.2013.61. 2

[BT88] BATTISTA G. D., TAMASSIA R.: Algorithms for plane representations of acyclic digraphs. *Theoretical Computer Science 61*, 2 (Nov. 1988), 175–198. doi:10.1016/0304-3975(88)90123-5. 3, 7

[BWC00] BHANIRAMKA P., WENGER R., CRAWFIS R.: Isosurfacing in higher dimensions. In *Proceedings of the conference on Visualization '00* (2000), IEEE Computer Society, pp. 267–273. doi:10.1109/VISUAL.2000.885704. 3

[CD14] CARR H., DUKE D.: Joint contour nets. *IEEE Transactions on Visualization and Computer Graphics 20*, 8 (Aug. 2014), 1100–1113. doi:10.1109/tvcg.2013.269. 3

[CLB11] CORREA C. D., LINDSTROM P., BREMER P.-T.: Topological spines: A structure-preserving visual representation of scalar fields. *IEEE Transactions on Visualization and Computer Graphics 17*, 6 (Aug. 2011), 1842–1851. doi:10.1109/TVCG.2011.244. 2

[CSA03] CARR H., SNOEYINK J., AXEN U.: Computing contour trees in all dimensions. *Computational Geometry 24*, 2 (Feb. 2003), 75–94. doi:10.1016/S0925-7721(02)00093-7. 2

[Dey06] DEY T. K.: *Curve and Surface Reconstruction: Algorithms with Mathematical Analysis*. Cambridge University Press, New York, NY, USA, 2006. doi:10.1017/CBO9780511546860. 3

[DPAM02] DEB K., PRATAP A., AGARWAL S., MEYARIVAN T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation 6*, 2 (Apr. 2002), 182–197. doi:10.1109/4235.996017. 3, 8, 10

[DTLZ02] DEB K., THIELE L., LAUMANNS M., ZITZLER E.: Scalable multi-objective optimization test problems. In *Proceedings of the 2002 Congress on Evolutionary Computation* (May 2002), vol. 1, IEEE, pp. 825–830. doi:10.1109/CEC.2002.1007032. 8, 10

[EKS83] EDELSBRUNNER H., KIRKPATRICK D. G., SEIDEL R.: On the shape of a set of points in the plane. *IEEE Transactions on Information Theory 29*, 4 (July 1983), 551–559. doi:10.1109/TIT.1983.1056714. 3, 8

[Far96] FARIN G. E.: *Curves and Surfaces for Computer-Aided Geometric Design: A Practical Code*, 4th ed. Academic Press, Inc., Orlando, FL, USA, 1996. 2, 7

[GAW*11] GLEICHER M., ALBERS D., WALKER R., JUSUFI I., HANSEN C. D., ROBERTS J. C.: Visual comparison for information visualization. *Information Visualization 10*, 4 (Sept. 2011), 289–309. URL: http://dx.doi.org/10.1177/1473871611416549, doi:10.1177/1473871611416549. 8

[GBPW10] GERBER S., BREMER P.-T., PASCUCCI V., WHITAKER R.: Visual exploration of high dimensional scalar functions. *IEEE Transactions on Visualization and Computer Graphics 16*, 6 (Nov./Dec. 2010), 1271–1280. doi:10.1109/TVCG.2010.213. 2

[GLG*13] GRATZL S., LEX A., GEHLENBORG N., PFISTER H., STREIT M.: LineUp: Visual analysis of multi-attribute rankings. *IEEE Transactions on Visualization and Computer Graphics 19*, 12 (Dec. 2013), 2277–2286. doi:10.1109/TVCG.2013.173. 3

[Han94] HANSON A. J.: *Geometry for n-dimensional graphics*. Academic Press Professional, Inc., San Diego, CA, USA, 1994, ch. Geometry for N-dimensional Graphics, pp. 149–170. URL: http://dl.acm.org/citation.cfm?id=180895.180909. 4

[HB03] HARROWER M. A., BREWER C. A.: ColorBrewer.org: An online tool for selecting color schemes for maps. *The Cartographic Journal 40*, 1 (June 2003), 27–37. doi:10.1179/000870403235002042. 6

[HC93] HANSON A. J., CROSS R. A.: Interactive visualization methods for four dimensions. In *IEEE Conference on Visualization, 1993* (Oct. 1993), IEEE, pp. 196–203. doi:10.1109/VISUAL.1993.398869. 3

[KDMW16] KIEFFER S., DWYER T., MARRIOTT K., WYBROW M.: HOLA: Human-like orthogonal network layout. *IEEE Transactions on Visualization and Computer Graphics 22*, 1 (Jan. 2016), 349–358. doi:10.1109/TVCG.2015.2467451. 3

[KLMA10] KARPENKO O., LI W., MITRA N. J., AGRAWALA M.: Exploded view diagrams of mathematical surfaces. *IEEE Transactions on Visualization and Computer Graphics 16*, 6 (Nov. 2010), 1311–1318. doi:10.1109/TVCG.2010.151. 3

[LBK04] LOTOV A. V., BUSHENKOV V. A., KAMENEV G. K.: *Interactive Decision Maps: Approximation and Visualization of Pareto Frontier*. Springer US, 2004. doi:10.1007/978-1-4419-8851-5. 2

[MBC79] MCKAY M. D., BECKMAN R. J., CONOVER W. J.: A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics 21*, 2 (May 1979), 239–245. doi:10.1080/00401706.1979.10489755. 5

[Mun09] MUNZNER T.: A nested model for visualization design and validation. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (Nov. 2009), 921–928. doi:10.1109/TVCG.2009.111. 2

[PBK10] PIRINGER H., BERGER W., KRASSER J.: HyperMoVal: Interactive visual validation of regression models for real-time simulation. *Computer Graphics Forum 29*, 3 (Aug. 2010), 983–992. doi:10.1111/j.1467-8659.2009.01684.x. 3

[Phi03] PHILLIPS G. M.: *Interpolation and Approximation by Polynomials*. CMS Books in Mathematics. Springer, New York, NY, 2003, ch. Bernstein polynomials, pp. 247–290. doi:10.1007/0-387-21682-0_7. 7

[PSTW*16] PAJER S., STREIT M., TORSNEY-WEIR T., SPECHTENHAUSER F., MÖLLER T., PIRINGER H.: WeightLifter: Visual weight space exploration for multi-criteria decision making. *IEEE Transactions on Visualization and Computer Graphics 23*, 1 (Jan. 2016), 611–620. doi:10.1109/TVCG.2016.2598589. 3

[SHB*14] SEDLMAIR M., HEINZL C., BRUCKNER S., PIRINGER H., MÖLLER T.: Visual parameter space analysis: A conceptual framework. *IEEE Transactions on Visualization and Computer Graphics 20*, 12 (Dec. 2014), 2161–2170. doi:10.1109/tvcg.2014.2346321. 2

[Shn96] SHNEIDERMAN B.: The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings of the 1996 IEEE Symposium on Visual Languages* (1996), pp. 336–343. doi:10.1109/VL.1996.545307. 2

[Sob67] SOBOL I. M.: On the distribution of points in a cube and the approximate evaluation of integrals. *USSR Computational Mathematics and Mathematical Physics 7*, 4 (1967), 86–112. doi:10.1016/0041-5553(67)90144-9. 6

[Som29] SOMMERVILLE D.: *Introduction to the Geometry of N Dimensions*. Dover Publications, 1929. 3, 7

[SWN03] SANTNER T. J., WILLIAMS B. J., NOTZ W. I.: *The Design and Analysis of Computer Experiments*. Springer Series in Statistics. New York, NY, USA, July 2003. doi:10.1007/978-1-4757-3799-8. 5

[TM04] TORY M., MÖLLER T.: Human factors in visualization research. *IEEE Transactions on Visualization and Computer Graphics 10*, 1 (Jan. 2004), 72–84. doi:10.1109/TVCG.2004.1260759. 2

[TS98] TWEEDIE L., SPENCE R.: The Prosection Matrix: A tool to support the interactive exploration of statistical models and data. *Computational Statistics 13*, 1 (Mar. 1998), 65–76. 2

[TWSM17] TORSNEY-WEIR T., SEDLMAIR M., MÖLLER T.: Sliceplorer: 1D slices for multi-dimensional continuous functions. *Computer Graphics Forum 36*, 3 (June 2017), 167–177. doi:10.1111/cgf.13177. 2, 3

[vWvL93] VAN WIJK J. J., VAN LIERE R.: HyperSlice: Visualization of scalar functions of many variables. In *Proceedings of the 4th Conference on Visualization* (Oct. 1993), IEEE Computer Society, pp. 119–125. doi:10.1109/VISUAL.1993.398859. 1, 2, 3

[Web17] WEBB R.: *Stella4D 5.4*, 2017. URL: http://www.software3d.com/Stella.php. 6, 7

[Wen13] WENGER R.: *Isosurfaces: Geometry, topology, and algorithms*. A.K. Peters/CRC Press, 2013. doi:10.1201/b15025. 3

[Zie01] ZIEGLER G. M.: Questions about polytopes. In *Mathematics Unlimited — 2001 and Beyond*. Springer Berlin Heidelberg, 2001, pp. 1195–1211. 7

[Zie12] ZIEGLER G. M.: *Lectures on polytopes*, vol. 152. Springer Science & Business Media, 2012. doi:10.1007/978-1-4613-8431-1. 2