Depth Explorer — A Software Tool for Analysis of Depth Measures

J. Hugg[*] E. Rafalin[*] D. L. Souvaine[*]

Abstract

We present Depth Explorer, a software tool for examination of depth measures. Depth Explorer was written in C++ and Objective-C. It uses a tree-based data structure to render 2-D statistical distributions and depth-based visualizations specified in XML files. We demonstrate the potential of the tool on several test cases. We present an experimental analysis of the L_1 depth measure that exposes its weaknesses. We use Depth Explorer to evaluate potential improvements to the L_1 depth measure.

1 Introduction

Multivariate data analysis has always been an important problem for statisticians. Recently, new analysis methods based on the Computational Geometry concept of Data Depth have been the focus of much research. While depth-based methods have proven very successful on many data sets, data depth is still a new concept and there are many research questions left to answer. There are many different definitions of data depth, and it has been shown that some problems are more suited to one definition than another. Comparing depth measures and choosing appropriate ones for specific problems is still a very difficult problem. Depth Explorer provides an experimental environment which can not only load pre-existing datasets but can generate random datasets from diverse distributions, can perform a variety of transformations on those data sets interactively, and can visualize the performance of the different depth measures under consideration on the disparate datasets.

The remainder of this paper is organized as follows. In Section 2, we present an overview of data depth and a few examples of the many possible ways to measure depth. In Section 3, we introduce **Depth Explorer**, a new, interactive tool for analyzing and comparing depth measures. With **Depth Explorer**, a researcher can evaluate a particular depth measure's performance on more data sets in less time than any other method. In Section 4, we use **Depth Explorer** to evaluate the strengths and weaknesses of a relatively new depth measure, L_1 depth. We propose a modification to L_1 depth and then use the tool to verify that the modification has positive effect. In Section 5, we discuss future directions for **Depth Explorer**.

2 Data Depth

Data depth is an analysis method that measures how central a point is relative to a cloud of data points. For example, given a cloud of data points in two dimensions, the most central point is called the median and this point has the highest depth value. Points on the periphery have the lowest depth values. Moving inward from the periphery to the median, the depth increases (See Figure 1).

[*]Department of Computer Science, Tufts University, Medford, MA 02155. {jhugg,erafalin,dls}@cs.tufts.edu. Partially supported by NSF grant CCF-0431027. Souvaine is a 2005-2006 MIT Visiting Scientist and Radcliffe Institute Fellow.

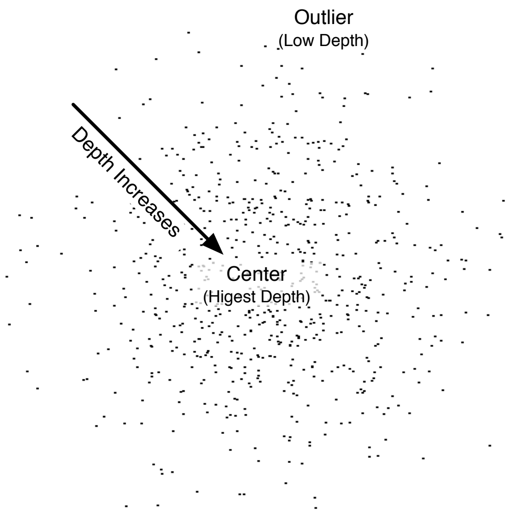


Figure 1: An Illustration of Data Depth

2.1 Examples of Depth Measures

Definition 1. The **convex-hull peeling depth** [5, 1] of a point X_k with respect to a data set $\mathcal{S} = \{X_1, \cdots X_n\}$ in $\mathbb{R}^d$ is the level of the convex layer to which X_k belongs.

The level of the convex layer is defined as follows: the points on the outer convex hull of $\mathcal{S}$ are designated level one and the points on the kth level are the points on the convex hull of the set $\mathcal{S}$ after the points on all previous levels were removed.

Points on the outer convex hull of a cloud will have depth 1. With those points, discarded, the next convex hull is found. Points on this secondary hull have depth 2. This continues until there are no points remaining.

The convex-hull peeling layers can be computed in the plane in optimal $\Theta(n \log n)$ time using Chazelle's deletions-only dynamic convex-hull algorithm [2] and in higher dimensions using iterative applications of any convex-hull algorithm (the complexity of computing the convex hull for a set of n points in $\mathbb{R}^d$ *once* is $O(n \log n + n^{\lfloor \frac{d}{2} \rfloor})$ [3, 13] and it can be applied as many as $O(n)$ times to compute the entire set of peeling layers).

Definition 2. The **half-space depth** [8, 16] (in the literature sometimes called *location depth* or *Tukey depth*) of a point or position x relative to a set of points $\mathcal{S} = \{X_1, ..., X_n\}$ is the minimum number of points of $\mathcal{S}$ lying in any closed half-space determined by a line through x (see Figure 2).

For all infinite lines through point x, the half-space depth is the minimum number of points on any one side of any line. Note that a point on the periphery will have depth zero, and a point at the center will have maximal depth. This technique can be expanded to work in n dimensions by considering the set of all hyperplanes that contain x.

The complete set of half-space depths of a set of points in $\mathbb{R}^d$ can be computed in $O(n^d)$ [15].

Definition 3. The **simplicial depth (SD)** [10] of a point x in $\mathbb{R}^d$ relative to a set of points $\mathcal{S} =$

$\{X_1, ..., X_n\}$ is the fraction of d simplicies[1] created by points of $\mathcal{S}$ that contain x.

$SD(\mathcal{S}; x) = \binom{n}{d+1}^{-1} \sum I_{(x \in S[X_{i_1}, ... X_{i_{d+1}}])}$, where I is the indicator function.

Consider all triangles that can be created from triples of points in the data cloud. The depth of x is the number of these triangles that contain x. Like half-space depth, simplicial depth is valid in higher dimensions. Unlike half-space depth, maximality at the center is not always a safe assumption.

It is possible to compute the simplicial depth of all n points in a cloud in $O(n^2)$ in $\mathbb{R}^2$ [6, 9], in $O(n^2)$ in $\mathbb{R}^3$ [6, 4, 14] and $O(n^4)$ in $\mathbb{R}^4$ [4]. In dimensions greater then 4, no better algorithm then the straightforward approach, requiring $O(n^{d+1})$, exists.

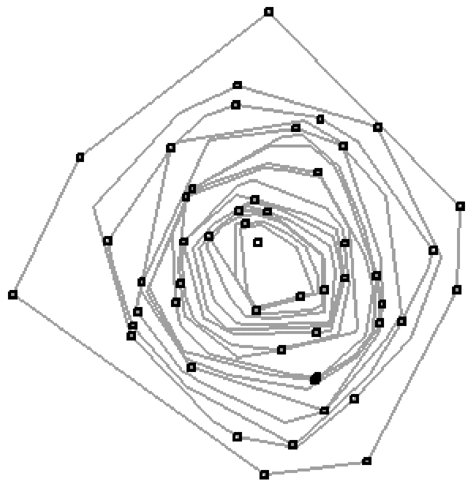


Figure 2: Contour lines along regions of equivalent half-space depth values

2.2 Depth Contours

Depth contours, nested contours that enclose regions of increasing depth, provide powerful tools to visualize and compare data sets. The contour of the **sample αth central region** is defined as the convex hull containing the most central fraction of α sample points [11] (see Figure 2). This contour is constructed by sorting all points of the original set according to their depth. A contour that encloses, for example, 75% of the points is created by taking the convex hull around data points $X_{[1]}, \cdots X_{[\lceil .75n \rceil]}$.

Depth contours help us visualize the *shape* of the data and determine outliers. Outliers are foreign influence on the data or errors. They can have an undesirable influence on the analysis of the data. The shape of a contour can be used to determine which data points might be outliers. One outlier detection method, proposed by Rousseeauw *et al.* [12], is to grow the 50% contour by a factor of three, and classify points outside the expanded region as outliers. This assumes that the depth measure used generates a 50% contour that is indicative of the overall shape of the data.

Note that depth contours as defined are only possible to compute on depth measures that satisfy the monotonicity property: As a point x moves away from the 'deepest point' along any fixed ray through the center, the depth at x should decrease monotonically [18]. Simplicial depth is an example of a depth measure that does not satisfy this property. Thus the convex hull of the deepest N% of

[1]A simplex in $\mathbb{R}^d$ is the boundary of the region defined by $d + 1$ vertices in general position, and represents the simplest possible polytope in any dimension. In one dimension, a simplex is simply a line segment; in 2 dimensions, it is a triangle; and in 3 dimensions, a tetrahedron.

points may contain more than N% of the points in the cloud. This makes simplicial depth a poor choice for generating depth contours.

2.3 Data Depth in High Dimensions

Presently, depth based statistics are proving very successful at analyzing bivariate data sets. While the concepts of depth and depth contours extend well to higher dimensions, depth-based tools are not as effective on multivariate data due to the poor scalability of the currently popular depth measures. The computational complexity of measures like half-space depth is exponentially slower in the number of dimensions. This means three-dimensional data will be slow to analyze. Analyzing ten dimensional data will be impossible.

Clearly, to use depth with high dimensional data, other depth measures or approximation schemes must be used. Rousseeauw has developed an approximation for the half-space depth measure that scales linearly in the number of dimensions [15]. A second scalable method is the L_1 depth measure developed by Vardi and Zhang [17].

Definition 4. The L_1 **depth** (L_1D) [17] of a point x with respect to a data set $\mathcal{S} = \{X_1, \cdots X_n\}$ in $\mathbb{R}^d$ is one minus the average of the unit vectors from x to all observations in $\mathcal{S}$:
$L_1D(S,x) = 1 - \|\bar{e}(x)\|$, where $e_i(x) = \frac{x - X_i}{\|x - X_i\|}$, $\bar{e}(x) = \frac{\sum_{i=1}^{n} \eta_i e_i(x)}{\sum_j \eta_j}$. η_i is a weight assigned to observation X_i (and is 1 if all observations are unique), and $\|x - X_i\|$ is the Euclidean distance between x and X_i.

Essentially, the L_1 median of a cloud of points in $\mathbb{R}^n$ is the point that minimizes the sum of the Euclidean distances to all points in the cloud. The L_1 depth of a point can be summarized by the question, "How much does the cloud need to be skewed before this point becomes the L_1 median?"

3 The Depth Explorer Statistical Sandbox

There are many ways to compare measure of depth. The statistics community has suggested desirable properties of depth functions and depth contours as a tool to analyze and evaluate depth functions [18, 7]. We focus on using an experimental method of analysis, rather then the theoretical method based on identifying desirable properties.

There is no definitive quantifiable way of comparing and evaluating depth measures. In fact most statistical tools can only be evaluated by comparing generated results from data sets. Most statistical software today focuses on using existing tools on fixed datasets. This facilitates the comparison of many tools on one particular set of data that is of interest to the user. To evaluate a particular tool abstractly, however, we must compare tools over the space of *all* datasets. While this complete approach is impractical, we can build a tool around this concept. We suggest the **Depth Explorer** statistical sandbox.

Depth Explorer allows for the automatic generation of data sets and the transformation or composition of existing datasets. For each data set, **Depth Explorer** can quickly visualize the behavior of the depth measure on the data. Rather than using static data and changing tools or the parameters to the tools, we have built software that fixes the tool settings, but focuses on perturbing the data. We present **Depth Explorer** as a software tool for examining statistical methods, specifically depth measures.

Depth Explorer is designed to be very simple to use. The user creates an XML text file describing a scene in any text editor. Data sets, transformations and visualizations are specified hierarchically using XML tags to describe a scene. We call this hierarchical representation of the scene the *render-tree*. After launching **Depth Explorer**, clicking one button selects the input file from the filesystem.

Clicking a second button renders the scene to a PDF file and displays that file in the window. Finally, the user can save the generated PDF file to the filesystem. While the program is running, the user can change the XML input file and click the render button again. The changes are almost instantly re-rendered into the window, allowing for interactive experimentation.

The **Depth Explorer** program is currently divided into two sections. A library called *libdepthengine* contains all of the code to load a *render-tree* into memory and render it to PDF. This library is written entirely in portable C++ but has two major dependencies. It currently uses PDFlib from GmbH Software[2] to generate PDF files and it uses Apple's implementation of BLAS[3] and LAPACK[4] to do matrix math and compute eigenvalues. PDFlib is open source very portable and thus does not introduce platform dependency. The BLAS and LAPACK routines used are standard and thus implemented in all implementations of BLAS and LAPACK, so this again, does not introduce platform dependence.

The second section is the **Depth Explorer** GUI. The GUI links against *libdepthengine* and is responsible for displaying, selecting input files, saving PDF renderings and printing. The GUI was developed with Objective-C using the Apple's Cocoa[5] application frameworks. Although the implementation is platform dependent, the amount of code is small and the complexity is very low compared to *libdepthengine*.

The speed of the rendering process allows **Depth Explorer** to give the user interactive feedback as the source data changes, however, even with the relatively fast depth measures like L_1, **Depth Explorer** does not give *instant* feedback. On a 1.5 Gigahertz Powerbook G4, rendering most scenes commonly requires 2-5 seconds. Rendering time is heavily dependent on the processing that is done in each node. If no depth-based visualizations are included in the render tree, than even the most complicated scene with thousands of data points will render almost instantly. When depth calculations are involved, computation time is slower. To render an L_1 50% depth contour on a cloud of 1000 points takes about two seconds. To render the same contour on 10,000 points requires about 5 seconds. While the scale-up should be linear according to the algorithm, we believe that matrix multiplication libraries are tuned for larger matrices and the speedup from using SIMD instructions is not fully realized until the data set gets larger.

Thus **Depth Explorer** is not a "sit and wait" program: almost all scenes can be rendered in a matter of seconds, not minutes. Conversely, **Depth Explorer** is not a real-time program: it is not fast enough to give dynamic feedback as the user changes parameters to the render tree nodes. Thus a render button is required. As more (slower) depth measures and complex visualizations are supported, it seems unlikely that **Depth Explorer** will ever become fully realtime. See the future work section for a discussion of speed improvements.

3.1 Representing Data Operations as a Tree

Depth Explorer renders data using an internal tree representation which it generates from the hierarchical XML input file. The leaf nodes must be data sources, either CSV files with a data set, or a random cloud generator with a set of points. **Depth Explorer** can currently create Gaussian clouds with any number of points with standard deviation 1, as well as uniformly distributed clouds on the range (-1,-1) to (1,1).

Non-leaf nodes must not contain new data, rather they should modify data or add visualizations to data. **Depth Explorer** supports affine transformation nodes that can scale, rotate or translate

[2]GmbH — http://www.pdflib.org/

[3]Basic Linear Algebra Subprograms — http://www.netlib.org/blas/

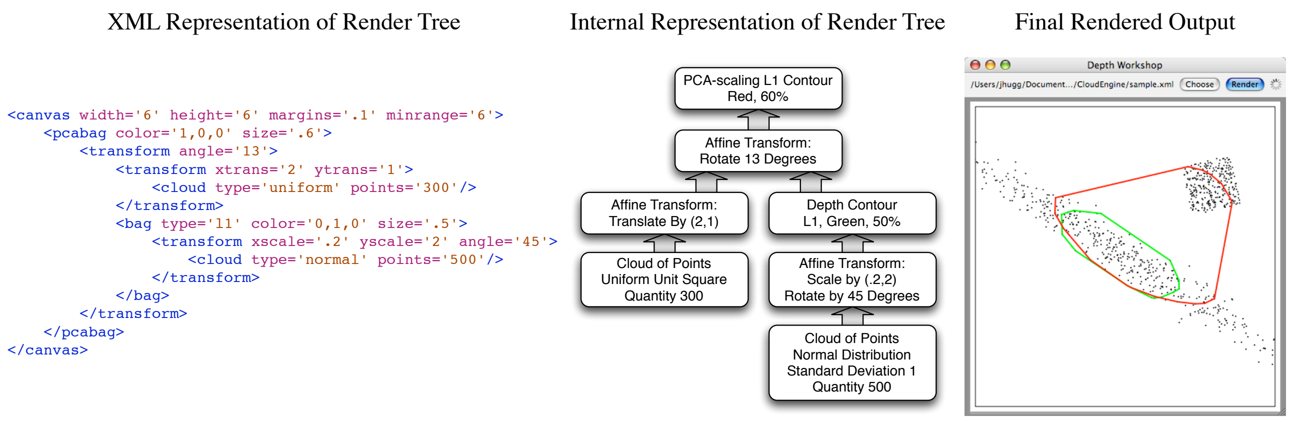
[4]Linear Algebra PACKage — http://www.netlib.org/lapack/

[5]Apple Cocoa — http://developer.apple.com/cocoa/

nodes below them. With affine transformations and any number of randomly generated clouds, a broad spectrum of data sets can be generated to test statistical methods.

Visualizations are visual implementations of statistical methods. To visualize a depth measure, contours can be drawn connecting points of equal depth. To visualize a clustering, points can be colored according to their clustering. As **Depth Explorer** uses a modular, tree-based renderer, it is trivial to both cluster data and generate depth contours at the same time, even at different points in the hierarchy.

The tree is rendered starting from the leaf nodes and moving up. Each node is a C++ module that, given the output of its children nodes, modifies or appends the data, then passes it to its parent. Ultimately, the root node, called the *canvas* node describes the dimensions and scale of the PDF output file and renders its childrens' data onto the PDF document.


XML Representation of Render Tree
Internal Representation of Render Tree
Final Rendered Output
PCA-scaling L1 Contour Red, 60%
Affine Transform: Rotate 13 Degrees
Affine Transform: Translate By (2,1)
Depth Contour L1, Green, 50%
Cloud of Points Uniform Unit Square Quantity 300
Affine Transform: Scale by (.2,2) Rotate by 45 Degrees
Cloud of Points Normal Distribution Standard Deviation 1 Quantity 500
Depth Workshop
/Users/jhugg/Document.../CloudEngine/sample.xml
Choose
Render

```xml
<canvas width='6' height='6' margins='.1' minrange='6'>
    <pcabag color='1,0,0' size='.6'>
        <transform angle='13'>
            <transform xtrans='2' ytrans='1'>
                <cloud type='uniform' points='300'/>
            </transform>
            <bag type='l1' color='0,1,0' size='.5'>
                <transform xscale='.2' yscale='2' angle='45'>
                    <cloud type='normal' points='500'/>
                </transform>
            </bag>
        </transform>
    </pcabag>
</canvas>
```

Figure 3: The Render tree form XML to Final Scene

The current version of **Depth Explorer** uses an XML markup to represent the render tree. XML is well suited due to its hierarchical nature, and many simple editors exist. The root of the XML tree and the render tree is the canvas tag. The canvas can describe a screen view, a page view or both. Sub-elements describe data, transformations and visualizations.

3.2 An Example of Using Depth Explorer

We often use depth contours to infer the shape of a data cloud. As our definition of depth contours mandates they must be convex, there are difficulties when computing depth contours on non-convex clouds. A convex shape becomes less and less accurate as the cloud becomes less convex. With **Depth Explorer**, it is a simple matter to specify a non-convex cloud and generate a depth contour. See Figure 4 for an example of a poorly fit contour.

4 Analysis of the L_1 Depth Measure using Depth Explorer

The L_1 depth measure is very attractive as it is very fast, simple to implement, and it is easy to parallelize. However, It has become clear, however, that the quality of results it generates is less than stellar. There are many different opinions on what constitutes a good depth measure, but the chief method of evaluation is using the depth measure in a calculation on real data and comparing the results to other measures.

Using the **Depth Explorer** tool, it is easy to test the L_1 depth measure's performance on many sample data sets quickly. As **Depth Explorer** currently supports only L_1 Depth and Convex Hull

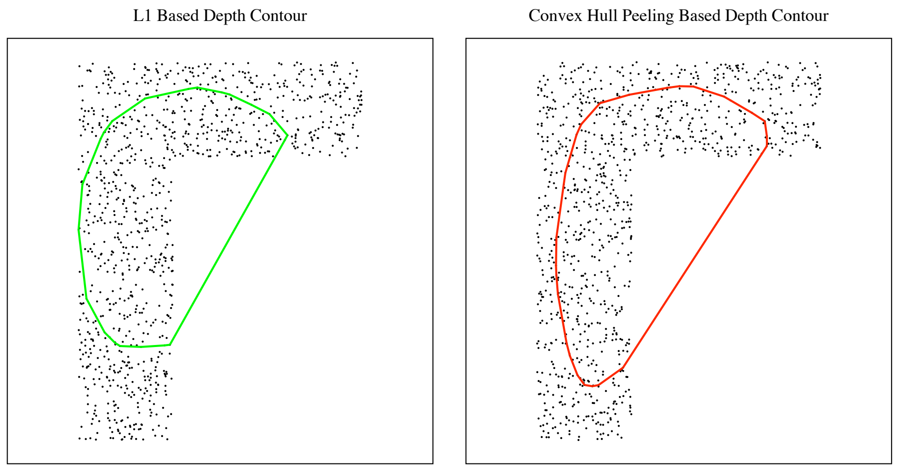


Figure 4: 50% Depth Contours in the L1 Depth Measure and the Convex Hull Peeling Depth Measure of a Non-Convex Cloud

Peeling Depth, peeling depth will be used as a reference for performance on simple data sets. It is certainly clear from Figure 5 that peeling depth gives a more representative shape of many data sets than L_1 depth.

Our experimentation lead to a primary observation about the L_1 depth: a point whose distance to the median point is small is much more likely to have higher depth than the same point in other depth measures. Consequently, L_1 depth contours tend to be much more circular than desired. In Figure 5, it is plain that a long and thin cloud of points has a much less thin 50% contour. Since the contour is supposed to be indicative of the shape of the cloud, the L_1 depth measure is not as useful as many other depth measures. Similarly, when the cloud is not round, as in the square uniform plot from Figure 5, the depth contours are still round, when they should be more rectangular to match the shape of the data.

One of the often cited qualities of a "good" depth measure is affine invariance. That is, the depth of a point in a cloud remains fixed under an affine transformation of the points. The L_1 depth measure is not affine invariant, but this may actually be helpful. As L_1 depth is most successful on hyper-spherical data, we suggest to perform affine transformations on non-spherical data to try to create a shape that approaches a hypersphere. The depth is computed on this scaled cloud, but the values are assigned to corresponding points in the original cloud.

The fundamental difficulty in a scaling L_1 depth measure, is determining the proper affine transformation to scale the data cloud to an approximate hyperspherical cloud. Our first attempt was to scale along the X and Y axis. This technique is unreliable. If the data cloud is compressed with respect to 45 or 135 degrees from the axis, the scaling has no effect. If the data is compressed with respect to an axis, the results are quite good. Essentially, an affine rotation of a few degrees can improve or degrade the results.

Again, we can use this weakness to our advantage. We use a technique called Principal Component Analysis (PCA) to determine the major directions of the data cloud. For d-dimensional data, PCA can generate a set of d perpendicular vectors who directions tell the general directions of the data cloud. Scaling along these vectors, rather than the axes, produces a much better approximation of a hyperspherical cloud. Like the original L_1 depth measure, *PCA-scaling-*L_1 depth is affine-rotation invariant. However, *PCA-scaling-*L_1 depth removes much of the aforementioned bias towards points closer to the median when assigning depth values.

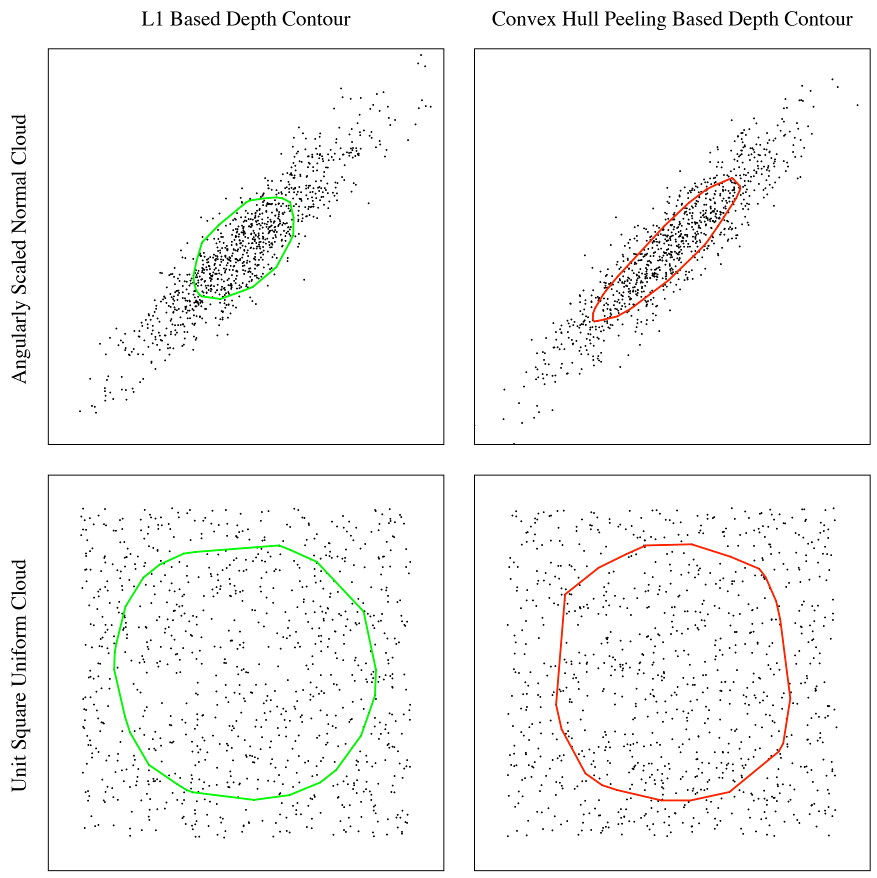


Figure 5: 50% Contours in the L1 Depth Measure and the Convex Hull Peeling Depth Measure

We used **Depth Explorer** to confirm this observation visually. See Figure 6 for an example of successful *PCA-scaling-*L_1 depth. Note that the final contour generated approximates the shape of the cloud. This new method of scaling does not, however, address the tendency for L_1 depth contours to be rounded in square data clouds as in Figure 5. Also note that this new method is still limited to handling data clouds that are roughly convex, but that again is a limitation of many depth measures.

In conclusion, we see no apparent reason to use L_1 depth instead of *PCA-scaling-*L_1 depth on any data cloud. *PCA-scaling-*L_1 depth seems to give more desirable results on any data set. **Depth Explorer** let us visually verify this on many generated data sets in a fraction of the time required by other methods. The time complexity of the calculation is dominated by the depth computation and *not* the PCA computation.

5 Future Work

Future work on **Depth Explorer** will have two chief directions. First, the **Depth Explorer** will be prepared for use by parties other than its authors. Primarily this means it must install and uninstall itself, it must support expected functionality like printing and it must contain online help. A secondary goal would be to support additional platforms other than Mac OS X. Like any project with a small development team, this must be weighed against reducing resources for other improvements to the tool. The primary hurdle to porting to windows at this point is the Apple provided BLAS libraries that are installed by default on every Macintosh computer. Porting to a Linux BLAS library may be

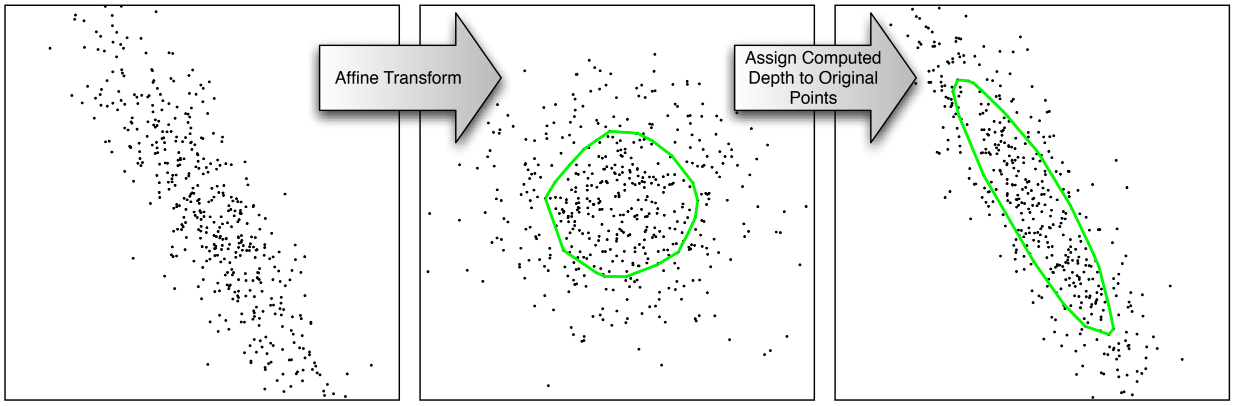


Figure 6: How Affine Scaling Improves the L_1 Depth Results

possible, but deployment may be difficult. Getting the tool running on Windows will require either time or money, as many commercial BLAS architectures exist, but free alternatives require more effort to use.

To make the tool more user friendly, it is a goal to move away from XML as a user interface, using it only as a file format for **Depth Explorer** scenes. There should be a tree-based graphical editor for specifying the render tree. This would protect against errors in the XML syntax, which currently cause the render to fail. It would also allow for flexible input methods. Instead of a numerical value, it could be possible to enter parameters to nodes with sliders, color-choosers, etc.. The tool suddenly becomes much more usable for the less technically savvy.

The second major direction for improvement to **Depth Explorer** is the addition of features. Presently, node-handlers that process nodes on the render tree are hardcoded into the application. This means additional node type functionality must be added by the maintainer. To aid collaboration and speed development, node-handlers should be moved to a defined plug-in architecture. Adding a new depth measure or visualization should be as simple as adding a file to the plug-ins folder for the application.

The number of visualizations can be increased to include much more than depth contours. Clustering visualizations could color points according to membership. Regions of depth could be shaded, rather than outlined. Points could display their numerical depth value as they are moused over.

Lastly the time required to render scenes can be reduced. The current tool is single-threaded, meaning it cannot take advantage of the recent push for dual core processors on the desktop. As the data is rendered in a tree from the bottom up, rendering subtrees in different threads would not be difficult. In fact, as the complexity of the scene increases, the parallelizability of the rendering does as well. We also plan to divide the render into two phases: the rendering of the data as points, followed by the rendering of the visualizations. As it is the depth calculations that take the largest share of the render time, it should be possible to allow real time modifications to the data, with updates to the visualizations lagging behind. It would also be helpful to have a progress bar showing the status of the render.

References

[1] V. Barnett. The ordering of multivariate data. *J. Roy. Statist. Soc. Ser. A*, 139(3):318–355, 1976.

[2] B. Chazelle. On the convex layers of a planar set. *IEEE Trans. Inform. Theory*, 31(4):509–517, 1985.

[3] B. Chazelle. An optimal convex hull algorithm in any fixed dimension. *Discrete Comput. Geom.*, 10(4):377–409, 1993.

[4] A. Y. Cheng and M. Ouyang. On algorithms for simplicial depth. In *Proc. 13th Canadian Conference on Computational Geometry*, pages 53–56, 2001.

[5] W. Eddy. Convex hull peeling. In H. Caussinus, editor, *COMPSTAT*, pages 42–47. Physica-Verlag, Wien, 1982.

[6] J. Gil, W. Steiger, and A. Wigderson. Geometric medians. *Discrete Math.*, 108(1-3), 1992. Topological, algebraical and combinatorial structures. Frolík's memorial volume.

[7] X. He and G. Wang. Convergence of depth contours for multivariate datasets. *Ann. Statist.*, 25(2):495–504, 1997.

[8] J. Hodges. A bivariate sign test. *The Annals of Mathematical Statistics*, 26:523–527, 1955.

[9] S. Khuller and J. S. B. Mitchell. On a triangle counting problem. *Inform. Process. Lett.*, 33(6):319–321, 1990.

[10] R. Liu. On a notion of data depth based on random simplices. *The Annals of Statistics*, 18:405–414, 1990.

[11] R. Liu, J. Parelius, and K. Singh. Multivariate analysis by data depth: descriptive statistics, graphics and inference. *The Annals of Statistics*, 27:783–858, 1999.

[12] I. R. Peter J. Rousseeuw and J. W. Tukey. The bagplot: A bivariate boxplot. *The American Statistician*, 53(4), 1999.

[13] F. P. Preparata and M. I. Shamos. *Computational geometry*. Texts and Monographs in Computer Science. Springer-Verlag, New York, 1985. An introduction.

[14] P. Rousseeuw and I. Ruts. Bivariate location depth. *Applied Statistics-Journal of the Royal Statistical Society Series C*, 45(4):516–526, 1996.

[15] P. J. Rousseeuw and A. Struyf. Computing location depth and regression depth in higher dimensions. *Statistics and Computing*, 8:193–203, 1998.

[16] J. W. Tukey. Mathematics and the picturing of data. In *Proc. of the Int. Cong. of Math. (Vancouver, B. C., 1974), Vol. 2*, pages 523–531. Canad. Math. Congress, Montreal, Que., 1975.

[17] Y. Vardi and C.-H. Zhang. The multivariate L_1-median and associated data depth. *Proc. Nat. Acad. Sci. USA.*, 97:1423–1426, 2000.

[18] Y. Zuo and R. Serfling. General notions of statistical depth function. *Ann. Statist.*, 28(2):461–482, 2000.

[19] Y. Zuo and R. Serfling. Structural properties and convergence results for contours of sample statistical depth functions. *Ann. Statist.*, 28(2):483–499, 2000.