Modern Signal Processing
MSRI Publications
Volume **46**, 2003

Fast X-Ray and Beamlet Transforms for Three-Dimensional Data

DAVID L. DONOHO AND OFER LEVI

Abstract. Three-dimensional volumetric data are becoming increasingly available in a wide range of scientific and technical disciplines. With the right tools, we can expect such data to yield valuable insights about many important phenomena in our three-dimensional world.

In this paper, we develop tools for the analysis of 3-D data which may contain structures built from lines, line segments, and filaments. These tools come in two main forms: (a) Monoscale: the X-ray transform, offering the collection of line integrals along a wide range of lines running through the image — at all different orientations and positions; and (b) Multiscale: the (3-D) beamlet transform, offering the collection of line integrals along line segments which, in addition to ranging through a wide collection of locations and positions, also occupy a wide range of scales.

We describe different strategies for computing these transforms and several basic applications, for example in finding faint structures buried in noisy data.

1. Introduction

In field after field, we are currently seeing new initiatives aimed at gathering large high-resolution three-dimensional datasets. While three-dimensional data have always been crucial to understanding the physical world we live in, this transition to ubiquitous 3-D data gathering seems novel. The driving force is undoubtedly the pervasive influence of increasing storage capacity and computer processing power, which affects our ability to create new 3-D measurement instruments, but which also makes it possible to analyze the massive volumes of data that inevitably result when 3-D data are being gathered.

Keywords: 3-D volumetric (raster-scan) data, 3-D x-ray transform, 3-D beamlet transform, line segment extraction, curve extraction, object extraction, linogram, slant stack, shearing, planogram.

Work partially supported by AFOSR MURI 95–P49620–96–1–0028, by NSF grants DMS 98–72890 (KDI), and by DARPA ACMP BAA 98-04.

As examples of such ongoing developments we can mention: Extragalactic Astronomy [50], where large-scale galaxy catalogs are being developed; Biological Imaging, where methods like single-particle electron microscopy and tomographic electron microscopy directly give 3-D data about structure of biological interest at the cellular level and below[45; 26]; and Experimental Particle Physics, where 3-D detectors lead to new types of experiments and new data analysis questions [22].

In this paper we describe tools which will be helpful for analyzing 3-D data when the features of interest are concentrated on lines, line segments, curves, and filaments. Such features can be contrasted to datasets where the objects of interest might be blobs or pointlike objects, or where the objects of interest might be sheets or planar objects. Effectively, we are classifying objects by their dimensionality; and for this paper the underlying objects of interest are of dimension 1 in R^3.

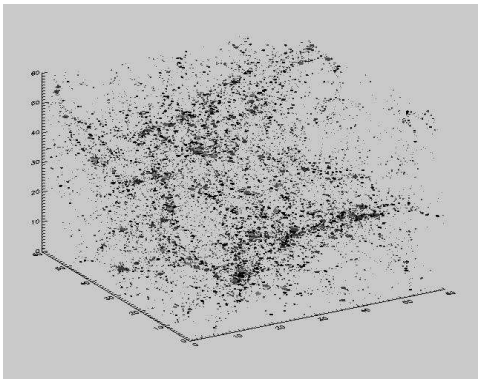

Figure 1. A simulated large-scale galaxy distribution. (Courtesy of Anatoly Klypin.)

1.1. Background motivation. As an example where such concerns arise, consider an exciting current development in extragalactic astronomy: the compilation and publication of the Sloan Digital Sky Survey, a catalog of galaxies which spans an order of magnitude greater scale than previous catalogs and which contains an order of magnitude more data.

The catalog is thought to be massive enough and detailed enough to shed considerable new light on the processes underlying the formation of matter and galaxies. It will be particularly interesting (for us) to better understand the filamentary and sheetlike structure in the large-scale galaxy distribution. This structure reflects gravitational processes which cause the matter in the universe to collapse from an initially fully three-dimensional scatter into a scatter concentrated on lower-dimensional structures [41; 25; 49; 48].

Figure 1 illustrates a point cloud dataset obtained from a simulation of galaxy formation. Even cursory visual inspection suggests the presence of filaments and perhaps sheets in the distribution of matter. Of course, this is artificial data. Similar figures can be prepared for real datasets such as the Las Campanas catalog, and, in the future, the Sloan Digital Sky Survey. To the eye, the simulated and real datasets will look similar. But can one say more? Can one rigorously compare the quantitative properties of real and simulated data? Existing techniques, based on two-point correlation functions, seem to provide only very weak ability to discriminate between various point configurations [41; 25].

This is a challenging problem, and we expect that it can be attacked using the methods suggested in this paper. These methods should be able to quantify the extent and nature of filamentary structure in such datasets, and to provide invariants to allow detailed comparisons of point clouds. While we do not have space to develop such a specific application in detail in this paper, we hope to briefly convey here to the reader a sense of the relevance of our methods.

What we will develop in this paper is a set of tools for digital 3-D data which implement the *X-Ray transform* and related transforms. For analysis of continuum functions $f(x, y, z)$ with $(x, y, z) \in \mathbf{R}^3$, the X-ray transform takes the form

$$(Xf)(L) = \int_L f(p)dp,$$

where L is a line in R^3, and p is a variable indexing points in the line; hence the mapping $f \mapsto Xf$ contains within it all line integrals of f.

It seems intuitively clear that the X-ray transform and related tools should be relevant to the analysis of data containing filamentary structure. For example, it seems that in integrating along any line which matches a filament closely over a long segment, we will get an unusually large coefficient, while on lines that miss filaments we will get small coefficients, and so the spread of coefficients across lines may reflect the presence of filaments.

This sort of intuitive thinking resembles what on a more formal level would be called the principle of matched filtering in signal detection theory. That principle says that to detect a signal in noisy data, when the signal is at unknown location but has a known signature template, we should integrate the noisy data against the signature template shifted to all locations where the signal may be residing. Now filaments intuitively resemble lines, so integration along lines is a kind of intuitive matched filtering for filaments. Once this is said, it becomes clear that one wants more than just integrating along lines, because filamentarity can be a relatively local property, while lines are global objects. As filaments might resemble lines only over moderate-length line segments, one might find it more informative to compare them with templates of line integrals over *line segments* at all lengths, locations, and orientations. Such segments may do a better job of matching templates built from fragments of the filament.

Hence, in addition to the X-ray transform, we also consider in this paper a multiscale digital X-ray transform which we call the beamlet transform. As defined here, the beamlet transform is designed for data in a digital $n \times n \times n$ array. Its intent is to offer multiscale, multiorientation line integration.

1.2. Connection to 2-D beamlets. Our point of view is an adaptation to the 3-D setting of the viewpoint of Donoho and Huo, who in [21] have considered beamlet analysis of 2-D images. They have shown that beamlets are connected with various image processing problems ranging from curve detection to image segmentation. In their classification, there are several levels to 2-D beamlet analysis:

- *Beamlet dictionary:* a special collection of line segments, deployed across orientations, locations, and scales in 2-D, to sample these in an efficient and complete manner.
- *Beamlet transform:* the result of obtaining line integrals of the image along all the beamlets.
- *Beamlet graph:* a graph structure underlying the 2-D beamlet dictionary which expresses notions of adjacency of beamlets. Network flow algorithms can use this graph to explore the space of curves in images very efficiently. Multiscale chains of 2-D beamlets can be expressed naturally as connected paths in the beamlet graph.
- *Beamlet algorithms:* algorithms for image processing which exploit the beamlet transform and perhaps also the beamlet graph.

They have built a wide collection of tools to operationalize this type of analysis for 2-D images. These are available over the internet [1; 2]. In the BeamLab environment, one can, for example, assemble the various components in the above picture to extract filaments from noisy data. This involves calculating beamlet transforms of the noisy data, using the resulting coefficient pyramid as input to processing algorithms which are organized around the beamlet graph and which use various graph-theoretical optimization procedures to find paths in the beamlet graph which optimize a statistical goodness-of-match criterion.

Exactly the same classification can be made in three dimensions, and very similar libraries of tools and algorithms can be built. Finally, many of the same applications from the two-dimensional case are relevant in 3-D. Our goal in this paper is to build the very basic components of this picture: describing the X-ray and beamlet transforms that we work with, the resulting beamlet pyramids, and a few resulting beamlet algorithms that are easy to implement in this framework. Unfortunately, in this paper we are unable to explore all the analogous beamlet-based algorithms — such as the algorithms for extracting filaments from noisy data using shortest-path and related algorithms in the beamlet graph. We simply scratch the surface.

1.3. Contents. The contents of the paper are as follows:

- Section 2 offers a discussion of two different systems of lines in 3-D, one system enumerating all line segments connecting pairs of voxel corners on the faces of the digital cube, and one system enumerating all possible slopes and intercepts.
- Section 3 discusses the construction of beamlets as a multiscale system based on these systems of lines, and some properties of such systems. The most important pair of properties being (a) the low cardinality of the system: it has $O(n^4)$ elements as opposed to the $O(n^6)$ cardinality of the system of all multiscale line segments, while (b) it is possible to express each line segment in terms of a short chain of $O(\log(n))$ beamlets.
- Section 4 discusses two digital X-ray transform algorithms based on the vertex-pairs family of lines.
- Section 5 discusses transform algorithms based on the slope-intercept family of lines.
- Section 6 exhibits some performance comparisons
- Section 7 offers some basic examples of X-ray analysis and synthesis.
- Section 8 discusses directions for future work.

2. Systems of Lines in 3-D

To implement a digital X-ray transform one needs to define structured families of digital lines. We use two specific systems here, which we call the vertex-pair system and the slope-intercept system. Alternative viewpoints on 'digital geometry' and 'discrete lines' are described in [33; 34].

2.1. Vertex-pair systems. Take an $n \times n \times n$ cube of unit volume voxels, and call the set of vertices V the voxel corners which are not interior to the cube. These vertices occur on the faces on the data cube, and there are about $6(n+1)^2$ such vertices. For an illustration, see Figure 2.

To keep track of vertices, we label them by the face they belong to $1 \le f \le 6$ and by the coordinates $[k_1, k_2]$ within the face.

Now consider the collection of all line segments generated by taking distinct pairs of vertices in V. This includes many 'global scale lines' crossing the cube from one face to another, at voxel-level resolution. In particular it does not contain any line segments with endpoints strictly inside the cube.

The set has roughly $18n^4$ elements, which can be usefully indexed by the *pair of faces* (f_1, f_2) they connect and the coordinates $[k_1^1, k_2^1], [k_1^2, k_2^2]$ of the endpoints on those faces. There are 15 such face-pairs involving distinct faces, and we can uniquely specify a line by picking any such face-pair and any pair of coordinate pairs obeying $k_i^j \in \{0, 1, 2, \ldots n\}$.

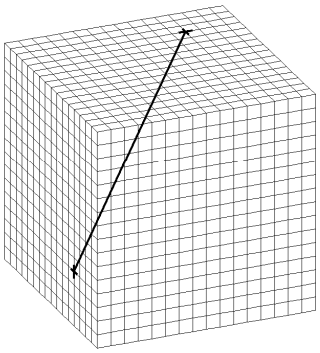

Figure 2. The vertices associated with the data cube are the voxel corners on the surface; a digital line indicated in red, with endpoints at vertices indicated in green.

2.2. Slope-intercept systems. We now consider a different family of lines, defined not by the endpoints, but by a parametrization. For this family, it is best to change the origin of the coordinate system so that the data cube becomes an $n \times n \times n$ collection of cubes with center of mass at $(0, 0, 0)$. Hence, for (x, y, z) in the data cube we have $|x|, |y|, |z| \leq n/2$. We can consider three kinds of lines: *x-driven, y-driven, and z-driven*, depending on which axis provides the shallowest slopes. An x-driven line takes the form

$$z = s_z x + t_z, \quad y = s_y x + t_y$$

with slopes s_z,s_y, and intercepts t_z and t_y. Here the slopes $|s_z|, |s_y| \leq 1$. y- and z-driven lines are defined with an interchange of roles between x and y or z, as the case may be.

We will consider the family of lines generated by this, where the slopes and intercepts run through an equispaced family:

$$s_x, s_y, s_z \in \{2\ell/n : \ell = -n/2, \ldots, n/2-1\}, \qquad t_x, t_y, t_z \in \{\ell : -n+1, \ldots, n-1\}.$$

3. Multiscale Systems: Beamlets

The systems of line segments we have just defined consist of *global scale* segments beginning and ending on faces of the cube. For analysis of fragments of lines and curves, it is useful to have access to line segments which begin and end well inside the cube and whose length is adjustable so that there are line segments of all lengths between voxel scale and global scale.

A seemingly natural candidate for such a collection is the family of *all* line segments between any voxel corner and any other voxel corner. For later use, we call such segments *3-D beams*. This set is expressive — it approximates any line segment we may be interested in to within less than the diameter of one voxel.

On the other hand, the set of all such beams can be of huge cardinality — with $O(n^3)$ choices for both endpoints, we get $O(n^6)$ 3-D beams — so that it is clearly infeasible to use the collection of 3-D beams as a basic data structure even for $n = 64$. Note that digital 3-D imagery is becoming available with $n = 2048$ from Resolution Sciences, Inc., Corte Madera, CA, and many important applications involve the analysis of volumetric images that contain filamentary objects such as blood vessel networks or fibers in a paper. For such datasets it seems natural to use beams-based analysis tools, however, working with $O(n^6)$ storage would be prohibitive.

The challenge, then, is to develop a *reduced-cardinality* substitute for the collection of 3-D beams, but one which is nevertheless expressive, in that it can be used for many of the same purposes as 3-D beams. Throughout this section we will be working in the context of vertex-pair systems of lines.

3.1. The beamlet system. A dyadic interval $D(j,k)$ satisfies $D(j,k) = [k/2^j, (k+1)/2^j] \subset [0,1]$ where k is an integer between 0 and 2^j; it has length 2^{-j}. A dyadic cube $C(k_1,k_2,k_3,j) \subset [0,1]^3$ is the direct product of dyadic intervals

$$[k_1/2^j,(k_1+1)/2^j] \otimes [k_2/2^j,(k_2+1)/2^j] \otimes [k_3/2^j,(k_3+1)/2^j]$$

where $0 \le k_1,k_2,k_3 < 2^j$ for an integer $j \ge 0$. Such cubes can be viewed as descended from the unit cube $C(0,0,0,0) = [0,1]^3$ by recursive partitioning. Hence, the splitting $C(0,0,0,0)$ in half along each axis $D(j,k_1) \otimes D(j,k_2) \otimes D(j,k_3)$ yields the eight cubes $C(k_1,k_2,k_3,1)$ where $k_i \in \{0,1\}$, splitting those in half along each axis we get the 64 subcubes $C(k_1,k_2,k_3,2)$ where $k_i \in \{0,1,2,3\}$, and if we decompose the unit cube into n^3 voxels using a uniform n-by-n-by-n grid with $n = 2^J$ dyadic, then the individual voxels are the n^3 cells $C(k_1,k_2,k_3,J)$, $0 \le k_1,k_2,k_3 < n$.

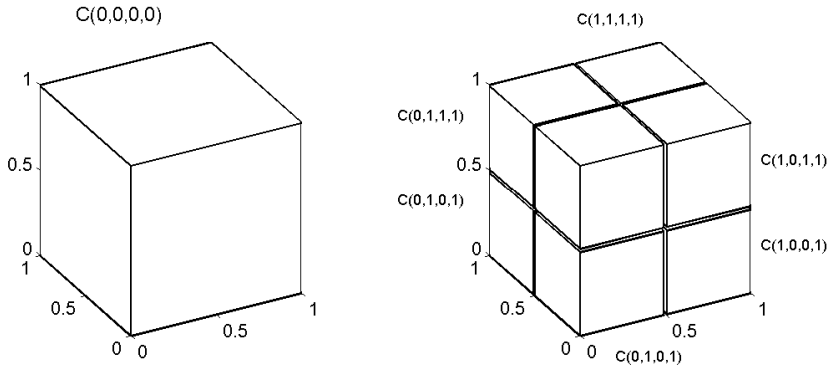

Figure 3. Dyadic cubes.

Associated to each dyadic cube we can build a system of lines based on vertex pairs. For a dyadic cube $Q = C(k_1, k_2, k_3, j)$ tiled by voxels of side $1/n$ for a dyadic $n = 2^J$ with $J > j$, let $V_n(Q)$ be the set of voxel corners on the faces of Q and let $B_n(Q)$ be the collection of all line segments generated by vertex-pairs from $V_n(Q)$.

DEFINITION 1. We call $B_n(Q)$ the set of *3-D beamlets* associated to the cube Q. Taking the collection of all dyadic cubes at all dyadic scales $0 \le j \le J$, and all beamlets generated by all these cubes, the *3-D beamlet dictionary* is the union of all the beamlet sets of all dyadic subcubes of the unit cube, and we denote this set by B_n.

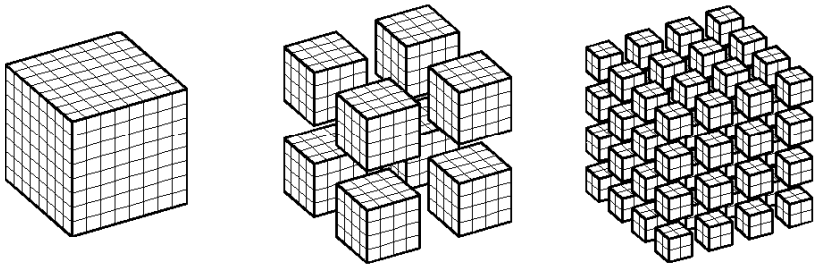

Figure 4. Vertices on dyadic cubes are always just the points on the faces of the cubes.

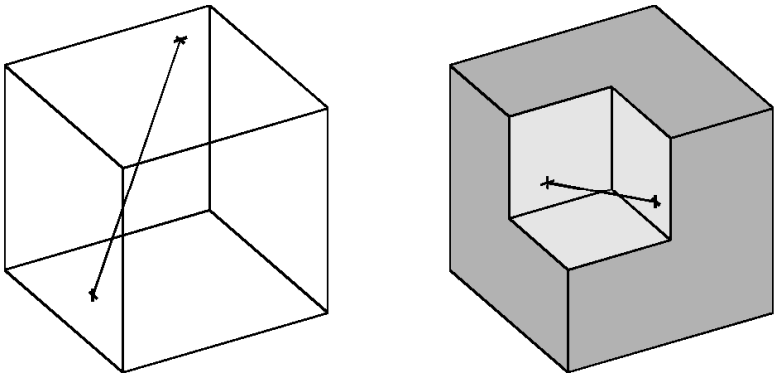

Figure 5. Examples of beamlets at two different scales: (a) scale 0 (coarsest scale); (b) scale 1 (next finer scale).

This dictionary of line segments has three desirable properties.

- It is a *multi-scale* structure: it consists of line segments occupying a range of scales, locations, and orientations.
- It has *controlled cardinality*: there are only $O(n^4)$ 3-D beamlets, as compared to $O(n^6)$ beams.

- It is *expressive*: a small number of beamlets can be chained together to approximately represent any beam.

The first property is obvious: the multi-scale, multi-orientation, multi- location nature has been obtained as a direct result of the construction.

To show the second property, we compute the cardinality of B_n. By assumption, our voxel size $1/n$ has $n = 2^J$, so there are $J + 1$ scales of dyadic cubes. Of course for any scale $0 \le j \le J$ there are 2^{3j} dyadic cubes of scale j; each of these dyadic cubes contains $2^{3(J-j)}$ voxels, approximately $6 \times 2^{2(J-j)}$ boundary vertices, and therefore $18 \times 2^{4(J-j)}$ 3-D beamlets.

The total number of 3-D beamlets at scale j is the number of dyadic cubes at scale j, times the number of beamlets of a dyadic cube at scale j, which gives $18 \times 2^{4J-j}$. Summing for all scales gives a total of approximately $36 \times 2^{4J} = O(n^4)$ elements total.

We will now turn to our third claim — that the collection of 3-D beamlets is expressive. To develop our support for this claim, we will first introduce some additional terminology and make some simple observations, and then state and prove a formal result.

3.2. Decompositions of beams into chains of beamlets. In decomposing a dyadic cube Q at scale j into its 8 disjoint dyadic subcubes at scale $j + 1$, we call those subcubes the children of Q, and say that Q is their parent. We also say that 2 dyadic cubes are siblings if they have the same parent. Terms such as descendants and ancestors have the obvious meanings. In this terminology, except at the coarsest and finest scales, all dyadic subcubes have 8 children, 7 siblings and 1 parent. The data cube has neither parents nor siblings and the individual voxels don't have children. We can view the inheritance structure of the set of dyadic cubes as a balanced tree where each node corresponds to a dyadic cube, the data cube corresponds to the root and the voxel cubes are the leaves. The depth of a node is simply the scale parameter j of the corresponding cube $C(k_1, k_2, k_3, j)$.

The dividing planes of a dyadic cube are the 3 planes that divide the cube into its 8 children; we refer to them as the x-divider, y-divider and z-divider. For example the x-divider of $C(0, 0, 0, 0)$ is the plane $\{(1/2, y, z) : 0 \le y, z \le 1\}$, the y-divider is $\{(x, 1/2, z) : 0 \le x, z \le 1\}$, and the z-divider is $\{(x, y, 1/2) : 0 \le x, y \le 1\}$.

We now make a remark about beamlets of data cubes at different dyadic n. Suppose we have two data cubes of sizes $n_1 = 2^{j_1}$ and $n_2 = 2^{j_2}$, and suppose that $n_2 > n_1$. Viewing the two data cubes as filling out the same volume $[0, 1]^3$, consider the beamlets in each system associated with a common dyadic cube $C(k_1, k_2, k_3, j)$, $0 \le j \le j_1 < j_2$. The collection of beamlets associated with the n_2-based system has a finer resolution than those associated with the n_2-based system; indeed every beamlet in the B_{n_1} also occurs in the B_{n_2}. Hence, in a natural sense, the beamlet families refine, and have a natural limit, B_∞,

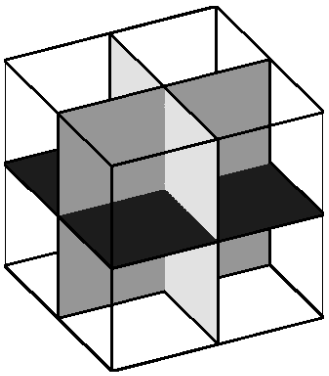

Figure 6. Dividing planes of a cube.

say. B_∞, of course, is the collection of all line segments in $[0,1]^3$ with both endpoints on the boundary of some dyadic cube. We will call members of this family the *continuum beamlets*, as opposed to the members of some B_n, which are *discrete beamlets*. Every discrete beamlet is also a continuum beamlet, but not the reverse.

LEMMA 1. Divide a continuum beamlet associated to a dyadic cube Q into the components lying in each of the child subcubes. There are either one, two, three or four distinct components, and these are continuum beamlets.

PROOF. Traverse the beamlet starting from one endpoint headed toward the other. If you travel through more than one subcube along the way, then at any crossing from one cube to another, you will have to penetrate one of the x-, y-, or z-dividers. You can cross each such dividing plane at most once, and so there can be at most 4 different subcubes traversed. □

THEOREM 1. Each line segment lying inside the unit cube can be approximated by a connected chain of m discrete beamlets in B_n where the Hausdorff distance from the chain to the beam is at most $1/n$ and where the number of links m in the chain is bounded above by $6log_2(n)$.

PROOF. Consider the arbitrary line segment ℓ inside the unit cube with endpoints v_1 and v_2 that are not necessary voxel corners. We can approximate ℓ with a beam b by replacing each endpoint with the closest voxel corner. Since the $\sqrt{3}/(2n)$ neighborhood of any point inside the unit cube must include a vertex, the Hausdorff distance between ℓ and b is bounded by $\sqrt{3}/(2n)$.

We now decompose the beam b into a minimal cardinality chain of connected continuum beamlets, by a recursive algorithm which starts with a line segment, and at each stage breaks it into a chain of continuum beamlets, with remainders on the ends, to which the process is recursively applied.

In detail, this works as follows. If b is already a continuum beamlet for $C(0,0,0,0)$ we are done; otherwise, b can be decomposed into a chain of (at most

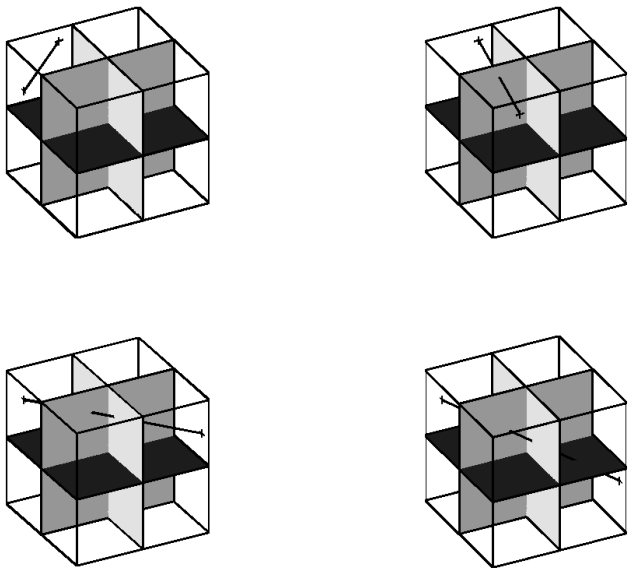

Figure 7. Decomposition of several beamlets into continuum beamlets at next finer scale, indicating cases which can occur.

four) segments based on crossings of b with the 3 dividing planes of $C(0,0,0,0)$. The interior segments of this chain all have endpoints on the dividing planes and hence are all continuum beamlets for the cubes at scale $j = 1$. We go to work on the remaining segments. Either endmost segment of the chain might be a continuum beamlet for the associated dyadic cube at scale $j = 1$; if so, we are done with that segment; if not, we decompose the segment into its components lying in the children dyadic cubes at scale $j = 2$. Again, the internal segments of this chain will be continuum beamlets, and additionally, at least one of the two endmost segments will be a continuum beamlet. If both endmost segments are continuum beamlets, then we are done. If not, take the segment which is not a beamlet and break it into its crossings with the dividing planes of the enclosing dyadic cube. Continue in this way until we reach the finest level, where, by hypothesis, we obtain a segment which has an endpoint in common with the original beam b. Since b is a beam, it ends in a vertex corner, and since the segment arose from earlier stages of the algorithm, the other endpoint is on the boundary of a dyadic cube. Hence the segment is a continuum beamlet and we are done.

Let's upperbound the number of beamlets generated by this algorithm. Assume always that we never fortuitously get an end segments to be a beamlet when it is not mandated by the above comments. So we have 2 continuum beamlets at the 1st scale and we are left with 2 segments to replace by 2 chains of discrete beamlets at finer scales. In the worst case, each of the segments when decomposed at the next scale, generates 3 continuum beamlets and 1 non-beamlet.

Continuing to the finest scale, in which the dyadic cubes are the individual voxels, we can have at most 2 beamlets in the chain at the finest scale. So in the worst case our chain will include 2 continuum beamlets at the 1st scale, 2 at the finest scale and 6 at any other scale $2, 3, ..., J-1$, So we get a maximum total of $2 + 6(J-1) + 2 = 6J - 2$ continuum beamlets needed to represent any line segment in the unit cube.

We now take the multiscale chain of beamlets and approximate it by a chain of discrete beamlets. The point is that the Hausdorff distance between line segments is upperbounded by the distance between corresponding endpoints. Now both endpoints of any continuum beamlet in B_∞ lie on certain voxel faces. Hence they lie within a $1/(\sqrt{2}n)$ neighborhood of some voxel corner. Hence any continuum beamlet in B_∞ can be approximated by a discrete beamlet in B_n within a Hausdorff distance of $1/(\sqrt{2}n)$. Notice that there may be several choices of such approximants; we can make the choice of approximant consistently from one beamlet to the next to maintain chain connectivity if we like.

So we get a maximum total of $6J - 2$ connected beamlets needed to approximate any line segment in the unit cube to within a Hausdorff distance of $\max\{\sqrt{3}/(2n), 1/(\sqrt{2}n)\} < 1/n$. □

The fact that arbitrary line segments can be approximated by relatively few beamlets implies that *every smooth curve can be approximated by relatively few beamlets*.

To see this, notice that a smooth curve can be approximated to within distance $1/m^2$ by a chain about m line segments — this is a simple application of calculus. But then, approximating each line segment in the chain by its own chain of $6\log(n)$ beamlets, we get approximation within distance $1/m^2 + 1/n$ by $O(\log(n) \cdot m)$ beamlets. Moreover, we can set up the process so that the individual chains of beamlets form a single unbroken chain. Compare also [17, Lemma 2.2, Corollary 2.3, Lemma 3.2].

4. Vertex-Pairs Transform Algorithms

Let $v = (k_1, k_2, k_3)$ be a voxel index, where $0 \le k_i < n$ and let $I(v)$ be the corresponding voxel intensities of a 3D digital image. Let $f(x)$ be the function on R^3 that represents the data cube by piecewise constant interpolation — i.e. the value $f(x) = I(v)$ when $x \in v$.

Definition 2. For each line segment $b \in B_n$, let $\gamma_b(\cdot)$ correspond to the unit speed path traversing b.

The discrete X-ray transform based on global-scale vertex-pairs lines is defined as follows. With $B_n([0,1])^3$ denoting the collection of vertex-pairs line segments of associated to the cube $[0,1]^3$,

$$X_I(b) = \int f(\gamma_b(\ell))\, \mathrm{d}\ell, \quad b \in B_n([0,1])^3.$$

The beamlet transform based on multiscale vertex-pairs lines is the collection of all multiscale line integrals

$$T_I(b) = \int f(\gamma_b(\ell))\,\mathrm{d}\ell, \quad b \in B_n.$$

4.1. Direct evaluation. There is an obvious algorithm for computing beamlet/X-ray coefficients: one at a time, simply compute the sums underlying the defining integrals. This algorithm steps systematically through the beamlet dictionary using the indexing method we described above, identifies the voxels on the path γ_b for each beamlet, visits each voxel and forms a sum weighting the voxel value with the arc length of γ_b in that voxel.

In detail, the sum we are referring to works as follows. Let $Q(v)$ denote the cube representing voxel v and γ_b the curve traversing b

$$T_I(b) = \sum I(v)\,\mathrm{Length}(\gamma_b \cap Q(v)).$$

Hence, defining weights $w_b(v) = \mathrm{Length}(\gamma_b(l) \cap Q(v))$ as the arc lengths of the corresponding fragments, one simply needs the sum $\sum_v w_b(v)I(v)$.

Of course, most voxels are not involved in this sum; one only wants to involve the voxels where $w_b > 0$. The straightforward way to do this, explicitly following the curve γ_b from voxel to voxel and calculating the arc length of the fragment of curve within the voxel, is inelegant and bulky. A far better way to do this is to identify three equispaced sequences and then merge them. Those sequences are: (1) the intersections of γ_b with the parallel planes $x = k_1/n$; (2) the intersections with the planes $y = k_2/n$; and (3) the intersections with the planes $z = k_3/n$. Each of these collections of intersections is equispaced and easy to calculate. It is also very easy to merge them in the order they would be encountered in a traverse of the beamlet in definite order. This merger produces the sequence of intersections that would be encountered if we pedantically tracked the progress of the beamlet voxel-by-voxel. The weights $w_b(v)$ are just the distances between successive points.

The complexity of this algorithm is rather stiff: on an $n \times n \times n$ voxel array there are order $O(n^4)$ beamlets to follow, and most of the sums require $O(n)$ flops, so the whole algorithm requires $O(n^5)$ flops in general. Experimental studies will be described below.

4.2. Two-scale recursion. There is an asymptotically much faster algorithm for 3-D X-ray and beamlet transforms, based on an idea which has been well-established in the two-dimensional case; see articles of Brandt and Dym [12], by Götze and Druckenmiller [29], and by Brady [9], or the discussion in [21].

The basis for the algorithm is the divide and conquer principle. As depicted in Figure 7, and proven in Lemma 1, each 3-D continuum beamlet can be de-

composed into 2, 3, or 4 continuum beamlets at the next finer scale:

$$b = \bigcup_i b_i \tag{4–1}$$

It follows that

$$\int f(\gamma_b(\ell))d\ell = \sum_i \int f(\gamma_{b_i}(\ell))d\ell.$$

This suggests that we build an algorithm on this principle, so that for $b \in B_n$ we identify several b_i associated to the child dyadic cubes of b, getting the formula

$$T_I(b) = \sum_i T_I(b_i).$$

Hence, if we could compute all the beamlet coefficients at the finest scale, we could then use this principle to work systematically from fine scales to coarse scales, and produce all the beamlet coefficients as a result.

The computational complexity of this fine-to-coarse strategy is obviously very favorable: it is bounded by $4B_n$ flops, since each coefficient's computation requires at most 4 additions. So we get an $O(n^4)$ rather than $O(n^5)$ algorithm.

There is a conceptual problem with implementing this principle, since in general, the decomposition of a discrete beamlet in B_n into its fragments at the next finer scale (as we have seen) produces continuum beamlets, i.e. the b_i are in general only in B_∞, and not B_n. Hence it is not really the case that the terms $T_I(b_i)$ are available from finer scale computations. To deal with this, one uses approximation, identifying discrete beamlets $\hat{b}_i$ which are 'near' the continuum beamlets, and approximates the $T_I(b_i)$ by combinations of 'nearby' $T_I(\hat{b}_i)$.

Hence, in the end, we get favorable computational complexity for an approximately correct answer. We also get one very large advantage: instead of computing just a single X-ray transform, it computes all the scales of the multiscale beamlet transform in one pass. In other words: it costs the same to compute all scales or to compute just the coarsest scale.

As we have described it, there are no parameters to 'play with' to control the accuracy, at perhaps greater computational expense. What to do if we want high accuracy? Staying within this framework, we can obtain higher precision by oversampling. We create an $N \times N \times N$ data cube, where $N = 2^e n$ where e is an oversampling parameter (e.g. e=3), fill the values from the original data cube by interpolation (e.g. piecewise constant interpolation), run the two-scale algorithm for B_N, and then keep only the coefficients associated to $b \in B_N \cap B_n$. The complexity goes up as 2^{4e}.

5. Slope-Intercept Transform Algorithms

We now develop two algorithms for X-ray transform based on the slope-angle family of lines described in Section 2.2. Both are decidedly more sophisticated than the vertex-pairs algorithms, which brings both benefits and costs.

5.1. The slant stack/shearing algorithm. The first algorithm we describe adapts a fast algorithm for the X-ray transform in dimension 2, using this as an 'engine', and repeatedly applying it to obtain a fast algorithm for the X-ray transform in dimension 3.

5.1.1. Slant Stack The fast slant stack algorithm has been developed by Averbuch et al. (2001) [6] as way to rapidly calculate all line integrals along lines in 2-dimensional slope/angle form; i.e. either x-driven 2-dimensional lines of the form

$$y = sx + t, -n/2 \le x < n/2;$$

where $s = k/n$ for $-n \le k < n$ and where $-n \le t < n$ or y-driven 2-dimensional lines of the form

$$x = sy + t, -n/2 \le y < n/2,$$

where s and t run through the same discrete ranges. The algorithm is approximate, because it does not exactly compute the voxel-level definition of X-ray coefficient assumed in Section 3 above (involving sums of voxel values times arc lengths). Instead, it computes exactly the appropriate sums deriving from so-called sinc-interpolation filters. For the set of x-driven lines we have

$$SlantStack(y = sx + t, I) = \sum_{u=-n/2}^{n/2-1} \tilde{I}(u, su + z),$$

where I is a 2D discrete array and $\tilde{I}$ is its 2D sinc interpolant. The transform for the y-driven lines is defined in a similar fashion with the roles of x and y interchanged. The algorithm can obtain approximate line integrals along all lines of these two forms in $O(n^2 \log(n))$ flops, which is excellent considering that the number of pixels is $O(n^2)$. It is achieved by using a discrete Projection-Slice theorem that relates the Slant Stack coefficients and the 2D Fourier coefficients. To be more specific, we are able to calculate the slant stack coefficients by first calculating the 2D Fourier Transform of I on a pseudopolar grid (see Figure 8) and then applying a series of 1-D inverse FFTs along radial lines. Each application of the 1-D inverse FFT yields a vector of coefficients that correspond to the slant-stack transform of I along a family of parallel lines.

Figure 9 shows backprojections of different delta sequences, each concentrated at a single point in the coefficient space and corresponding to a choice of slope-intercept pair. The panels show the 2-D arrays of weights involved in the coefficient computation. Summing with these weights is approximately the same as exactly summing along lines of given slope/intercept.

As Averbuch et al. point out, the fast slant stack belongs to a group of algorithms developed over the years in synthetic aperture radar by Lawton [40] and in medical imaging by Pasciak [44] and by Edholm and Herman [24], where it is called the Linogram. The Linogram has been exploited systematically for more than ten years in connection with many problems of medical imaging, including

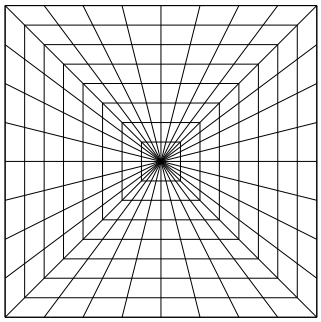

Figure 8. The Pseudopolar Grid is constructed from concentric squares $n = 8$ are converted into data at the intersections of concentric squares and lines radiating from the origin with equispaced slopes.

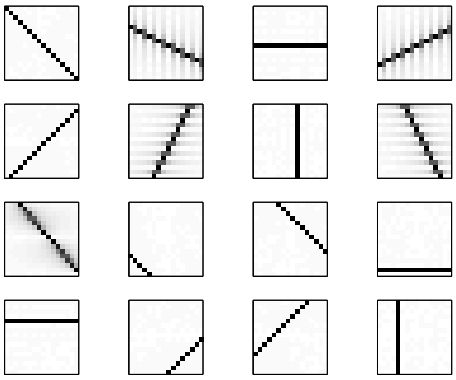

Figure 9. 2D Slant Stack Lines.

cone-beam and fan-beam tomography, which concern image reconstruction from subsets of the X-ray transform. In a 3-D context the most closely related work in medical imaging concerns the *planogram*; see [38; 39], and our discussion in Section 10.5 below. The terminology 'slant stack' comes from seismology, where this type of transform, with different algorithms, has been in use since the 1970's [15].

5.1.2. Overall Strategy We can use the slant stack to build a 3-D X-ray transform by grouping together lines into subfamilies which live in a common plane. We then extract that plane from the data cube and apply the slant stack to that plane, rapidly obtaining integrals along all lines in that plane. We ignore for the moment the question of how to extract planes from digital data when the planes are not oriented along the coordinate axes.

In detail, our strategy works as follows. Suppose we want to get transform coefficients corresponding to x-driven 3-D lines, i.e. lines obeying

$$y = s_y x + t_y, \quad z = s_z x + t_z.$$

Within the family of all n^4 lines of this type, consider the subfamily $\mathcal{L}_{xz,n}(s_z, t_z)$ of all lines with a fixed value of (s_z, t_z) and a variable value of (s_y, t_y). Such lines all lie in the plane $P_{xz}(s_z, t_z)$ of (x, y, z) with (x, y) arbitrary, $z = s_z x + t_z$. We can consider this set of lines as taking all x-driven 2-D lines in the (x, y) plane and then 'tilting' the plane to obey the equation $z = s_z x + t_z$. Our intention is to extract this plane, sampling it as a function of x and y, and use the slant stack to evaluate all the line integrals for all the x-driven lines in that plane, thereby obtaining all the integrals in $\mathcal{L}_{xz,n}(s_z, t_z)$ at once, and to repeat this for other families, working systematically through values of s_z and t_z.

Some of these subfamilies with constant intercept t and varying slope s are depicted in Figure 10.

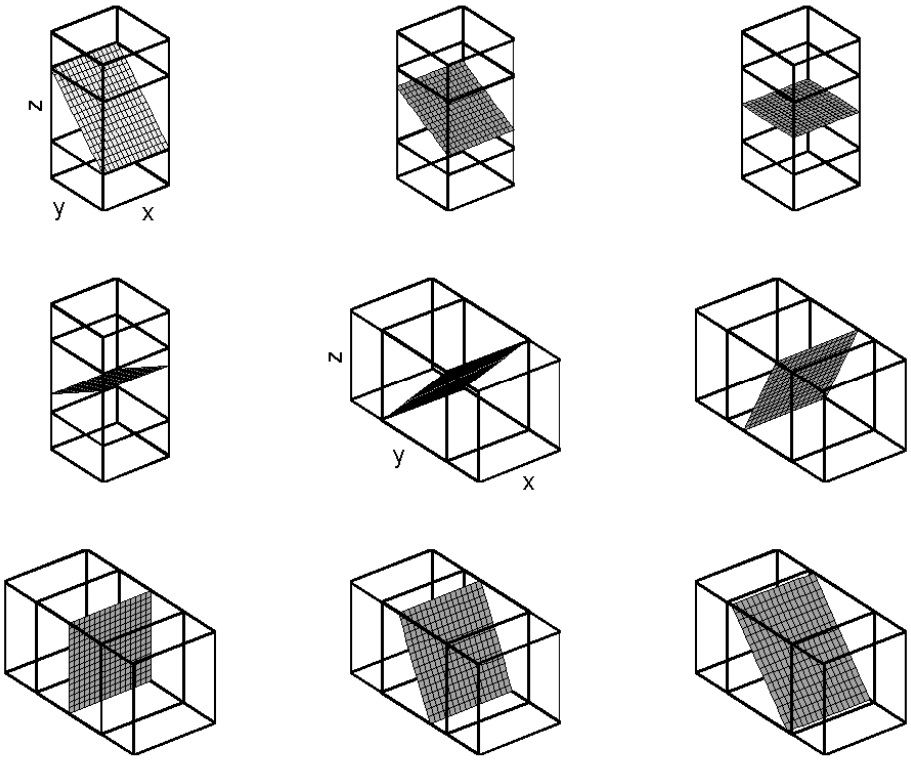

Figure 10. Planes generated by families of lines in the Slope-Angle dictionary; subpanels indicate various choices of slope.

In the end, then, our coordinate system for lines has one slope and one intercept to specify a plane and one slope and one intercept to specify a line within the plane.

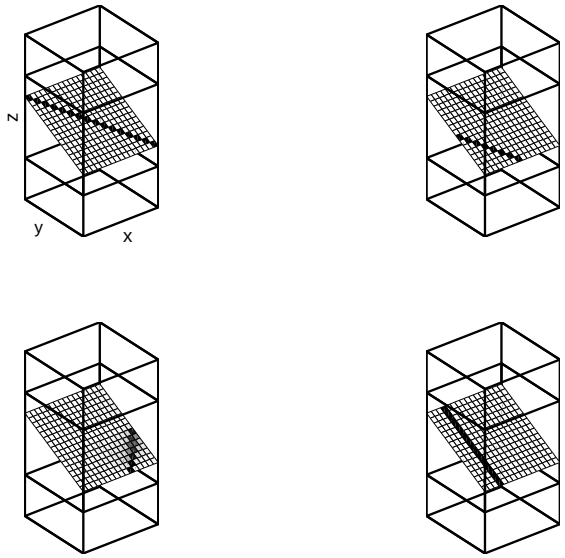

Figure 11. Lines selected from planes via slope-intercept indexing.

5.1.3. 3-D Shearing To carry out this strategy, we need to extract data lying in a general 2-D plane within a digital 3-D array.

We make a simple observation: to extract from the function $f(x, y, z)$ defined on the full cube its restriction to the plane with $z = s_z x + t_z$, and x, y varying, we simply create a new function $f'(x, y, z)$ defined by

$$f'(x, y, z) = f(x, y, z - s_z x - t_z)$$

for x, y, z varying throughout $[0, 1]^3$, with f taken as vanishing at arguments outside the unit cube. We then take $g(x, y) = f'(x, y, 0)$ as our extracted plane. The idea is illustrated in Figure 12.

In order to apply this idea to the case of digital arrays $I(x, y, z)$ defined on a discrete grid, note that, in general, $z - s_z x - t_z$ will not be an integer even when z and x are, and so the expression $I(x, y, z - s_z x - t_z)$ is not defined; one needs to make sense of this quantity somehow. At this point we invoke the notion of *shearing of digital images* as discussed, for example, in [54; 6]. Given a 2-D $n \times n$ image $I(x, y)$ where $-n/2 \le x, y < n/2$, we define the shearing of y as a function x at slope s, $Sh_{xy}^{(s)}$, according to

$$(Sh_{xy}^{(s)} I)(x, y) = I_2(x, y - sx).$$

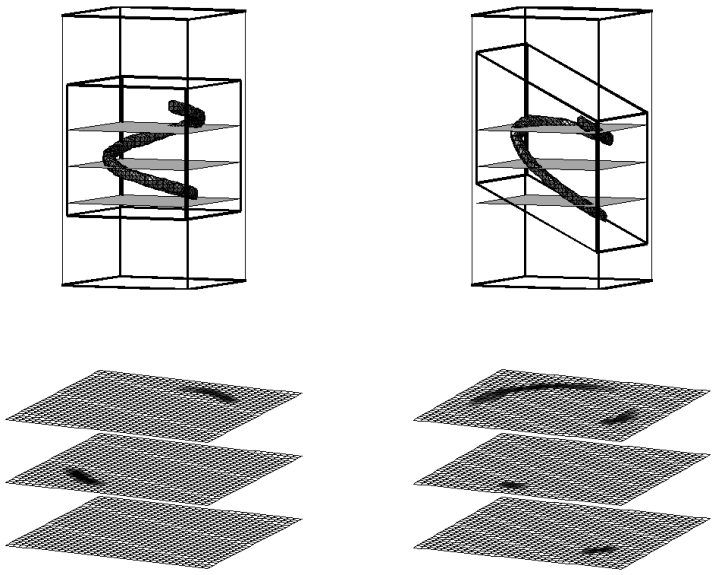

Figure 12. Shearing and slicing a 3D image. Extracting horizontal slices of a sheared 3-D image is the same as extracting slanted slices of the original image.

In words, the image is shifted vertically in each column x =constant, with the shift varying from one column to the next in an x-dependent way. Here $I_2(x, y)$ is an image which has been interpolated in the vertical direction so that the second argument can be a general real number and not just an integer. Specifically,

$$I_2(x, u) = \sum_v \phi_n(u - v) I(x, v),$$

where ϕ_n is an interpolation kernel — a continuous function of a real variable obeying $\phi_n(0) = 1$, $\phi_n(k) = 0$ for $k \neq 0$. The shearing of x as a function of y works similarly, with

$$(Sh_{yx}^{(s)} I)(x, y) = I_1(x - sy, y),$$

with

$$I_1(u, y) = \sum_v \phi_n(u - v) I(v, y).$$

We define a shearing operator for a 3-D data cube by applying a 2-D operator systematically to each 2-D planes in a family of parallel planes normal to one of the coordinate axes. Thus, if we speak of shearing in z as a function of x, we mean

$$Sh_{xz}^{(s)} I(x, y, z) = I_3(x, y, z - sx).$$

What shearing does is map a family of tilted parallel planes into a plane normal to one of the coordinate axes. In the above example, data along the plane $z = sx + t$ is mapped onto the plane $z = t$. Figure 12 illustrates the process

graphically, exaggerating the process, by allowing pieces of the original image to be sheared out of the original data volume. In fact those pieces 'moving out' of the data volume get 'chopped away' in actual computations.

5.1.4. The Algorithm Armed with this tool, we define the slant stack based X-ray transform algorithm as follows, giving details only for a part of the computation. The algorithm works separately with x-driven, y-driven, and z-driven lines. The procedure for x-driven lines is as follows:

- for each slope s_z
  - Shear z as a function of x with slope s_z, producing the 3-D voxel array I_{xz,s_z}.
  - for each intercept t_z
    - Extract the 2-D image $I_{s_z,t_z}(x,y) = I_{xz,s_z}(x,y,t_z)$.
    - Calculate the 2-D X-ray transform of this image, obtaining an array of coefficients $X(s_y,t_y)$, and storing these in the array $X_3('x', s_y, t_y, s_z, t_z)$.
  - end for
- end for

The procedure is analogous for y- and z- driven lines.

The lines generated by this algorithm are as illustrated in Figure 11.

The time complexity of this algorithm is $O(n^4 \log(n))$. Indeed, the cost of the 2-D slant-stack algorithm is order $n^2 \log(n)$ (see [6]), and this must be applied order n^2 times, one for each member of $\mathcal{L}_{xz,n}(s_z,t_z)$

5.2. Compatibility with cache memory. A particularly nice property of this algorithm is that it is *cache-aware* , i.e. it is very well-organized for use with modern hierarchical memory computers [32]. In currently dominant computer architectures, main memory is accessed at a speed which can be an order of magnitude slower than the cache memory on the CPU chip. As a result, other things being equal, an algorithm runs much faster if it operates as follows:

- Load n items from main memory into the cache
- Work intensively to compute n results
- Send the n results out to main memory

Here the idea is that the main computations involve relatively small blocks of data that can be kept in cache all at once, are referred to many times while in the fast cache memory, saving dramatically on main memory accesses.

The Slant-Stack/Shearing algorithm we have described above has exactly this form. In fact it can be decomposed in steps, every one of which can be conceptualized as follows:

- Load n items from main memory into the cache
- Do some combination of:

  - Compute an n-point forward FFT ; or
  - Compute an n-point inverse FFT ; or
  - Perform elementwise transformation on the n-vector;
- Send the n results out to main memory

Thus the 2-D slant stack and the 3-D data shearing operations can all be decomposed into steps of this form. For example, data shearing requires computing sums of the form $I'(x,y,z) = \sum_u \phi(z - sx - u)I(x,y,u)$. For each fixed (x,y), we take the n numbers $(I(x,y,u) : u = -n/2, ..., n/2 - 1)$, take their 1-D FFT along the last slice, multiply the FFT by a series of appropriate coefficients, and then take their inverse 1-D. The story for the slant stack is similar, but far more complicated. A typical step in that algorithm involves the 2-D FFT, which is obtained by applying order $2n$ 1-D FFT's, once along each row and once along each column. For more details see comments in [6].

It is also worth remarking that several modern CPU architectures offer FFT *in silico*, so that the FFT step in the above decomposition runs without any memory accesses for instruction fetches. Such architectures (which include the G4 processor running on Apple Macintosh and IBM RS/6000) are even more favorable towards this algorithm.

As a result of this cache- and CPU-favorable organization the observed behavior of this algorithm is far more favorable than what asymptotic theory would suggest. The vertex-pairs algorithms of the previous section sit at the opposite extreme; since those algorithms involve summing data values along lines, and the indices of those values are scattered throughout the linear storage allocated to the data cube, those algorithms appear to be performing essentially random access to memory; hence such algorithms run at the memory access speed rather than the cache speed. In some circumstances those algorithms can even run more slowly still, since cache misses can cost considerably more than one memory access, and random accesses can cause large numbers of cache misses. These remarks are in line with behavior we will observe empirically below.

5.3. Frequency domain algorithm. Mathematical analysis shows that the 3-D X-ray transform of a continuum function $f(x,y,z)$ can be obtained from the Fourier transform [51; 47]. This frequency-domain approach requires coordinatizing planes through the origin in frequency space by

$$\mathcal{P}_{\mathbf{u}_1,\mathbf{u}_2} = \{\xi = \mathbf{u}_1\xi_1 + \mathbf{u}_2\xi_2\}$$

extracting sections of the Fourier transform along such planes,

$$\hat{g}(\xi_1,\xi_2) = \hat{f}(\mathbf{u}_1\xi_1 + \mathbf{u}_2\xi_2),$$

and then taking the inverse Fourier transform of those sections:

$$g = \mathcal{F}^{-1}\hat{g}.$$

The resulting function g gives the X-ray transform for lines

$$g(x_1, x_2) = \int f(x_1\mathbf{v}_1 + x_2\mathbf{v}_2 + t\mathbf{v}_3)dt,$$

with an appropriate orthobasis $(\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)$.

To carry this out with digital data would require developing a method to efficiently extract many planes through the origin of the Fourier transform cube, and then perform 2-D inverse FFT's of the data in those planes. But how to rapidly extract a rich selection of planes through the origin? (The problem initially sounds similar to the problem encountered in the previous section, but recall that the set of planes needed there were families of parallel planes, not families of planes through the origin.

Our approach is as follows. Pick a fixed preferred coordinate axis, x, say. Pick a subordinate axis, z, say. In each constant-y slice, do a two-dimensional shearing of the FT data, shearing z as a function of x at fixed slope s_z. In effect, we have tilted the data cube, so that slices normal to the z-axis in the sheared volume correspond to tilted planar slices in the original volume. So now take each y-z plane, and apply idea of Cartesian-to-pseudopolar conversion as described in [6]. This uses interpolation to convert a planar Cartesian grid into a new point set consisting of n lines through the origin at various angles, and equispaced samples along each line. This conversion being done for each plane with x fixed, then, grouping the data in a given line through the origin across all x values produces a plane; see Figure 13. We then take a 2-D inverse transform of the data in this plane.

The computational complexity of the method goes as follows. $O(n^3 \log(n))$ operations are required for transforming from the original space domain to the frequency domain; $O(n^2 \log(n))$ work for each conversion of a Cartesian plane to

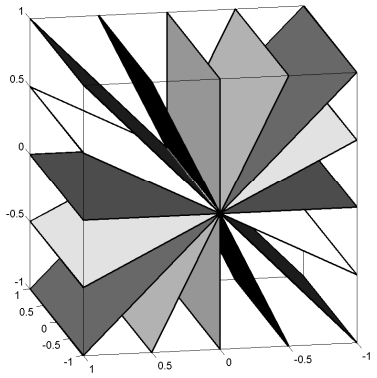

Figure 13. Selecting planes through the origin. Performing cartesian-to-pseudo-polar conversion in the yz plane and then gathering all the data for one radial line across different values of x produces a series of planes through the origin.

pseudopolar coordinates, giving $O(n^3 \log(n))$ work to convert a whole stack of parallel planes in this way; $O(n^3 \log(n))$ work to shear the array as a function of the preferred coordinate; and $3n$ such shearings need to be performed. Overall, we get $O(n^4)$ coefficients in $O(n^4 \log(n))$ flops.

We have not pursued this method in detail, for one reason: *it is mathematically equivalent to the slant-stack-and-shearing algorithm*, providing exactly the same results (assuming exact arithmetic). This is a consequence of the projection-slice theorem for the slant stack transform proved in [6].

6. Performance Measures

We now consider two key measures of performance of the fast algorithms just defined: accuracy and timing.

6.1. Accuracy of two-scale recursion. To estimate the accuracy of the two-scale recursion algorithm, we considered a 16^3 array and compared coefficients from two-scale approximation with direct evaluation. We computed the average error for the different scales and applied the algorithms both to a 3-D image that contains a single beamlet and to a 3-D image that contains randomly distributed ones in a sea of zero, chose so that both 3D images has the same $l2$ norm. The table below shows that the coefficients obtained from the two-scale recursion are significantly different from those of direct evaluation.

Analyze Single Beamlet		**Analyze Random Scatter**	
scale	relative error	scale	relative error
0	0.117	0	0.056
1	0.107	1	0.061
2	0.076	0	0.048
3	1.5×10^{-17}	3	3.7×10^{-17}

One way to understand this phenomenon is to look at what the coefficients are measuring by studying the *equivalent kernels* for those coefficients. Let T^1 be the linear transform on I corresponding to the exact evaluation of the line integrals and let T^2 be the linear transform corresponding to the two-scale recursion algorithm. Apply the adjoint of each transform to a coefficient-space vector with a one in one position and a zero in other positions, getting

$$w_b^j = (T^j)' \delta^b, \quad j = 1, 2. \tag{6–1}$$

Each w_b^j lives in image-space — i.e., it is indexed by voxels v, and the entries $w_b(v)$ indicate the weights such that $T_I[b] = \sum_v I(v) w_b(v)$. In essence this 'is' the beamlet we are using in that beamlet transform. For later use: we call the operation of calculating w_b that of 'backprojection', because we are going back from coefficient space to image space. This usage is consistent with usage of the term in the tomographic literature, i.e. [47; 15].

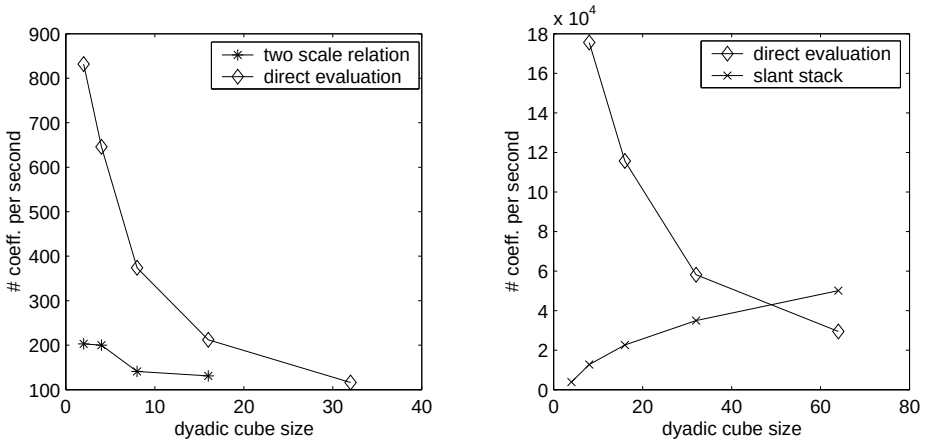

Figure 14. Timing comparison.

6.2. Timing comparison. The defining feature of 3-D processing is the massive volume of data involved and the attendant long execution times for even basic tasks. So the burning issue is: how do the algorithms perform in terms of CPU time to complete the task? The display in Figure 14 below shows that both the direct evaluation and the two scale recursion methods slow down dramatically as n increases — one expects a $1/n^{5/3}$ or $1/n^{4/3}$ scaling law to be evident in this display, and in rough terms, the display is entirely consistent with that law. The surprising thing in this display is the *improvement* in performance of the slant stack with increasing n. This seeming anomaly is best interpreted in terms of the cache-awareness of the slant stack algorithm. The slant stack algorithm becomes more and more immune to cache misses as n increases (at least in the range we are studying), and so the number of cache misses per coefficient drops lower and lower for this algorithm, while this effect is totally absent for the direct evaluation and two-scale recursion algorithm.

7. Examples of X-Ray Transforms

We now give a few examples of the X-ray transform based on the slant stack method.

7.1. Synthesis. While we have not discussed it at length, the adjoint of the X-ray transform is a very useful operator; for each variant of the X-ray transform that we have discussed, the corresponding adjoint can be computed using ideas very similar to those which allowed to compute the transform itself, and with comparable computational complexity. Just as the X-ray transform takes voxel arrays into X-ray coefficient arrays, the adjoint transform takes X-ray coefficient arrays into voxel arrays.

We have already mentioned, near (6–1) above, that when the adjoint operator is applied to a coefficient array filled with zeros except for a one in a single slot, the result is a voxel array. This array contains the weights $w_b(v)$ underlying the corresponding X-ray transform coefficient. In formal mathematical language this is the Riesz representer of the b-th coefficient. Intuitively, the representer should have its nonzero weights all concentrated on or near the corresponding 'geometrically correct' line.

To check this, we depict in Figure 15 representers of four different X-ray coefficients. Evidently, these are geometrically correct.

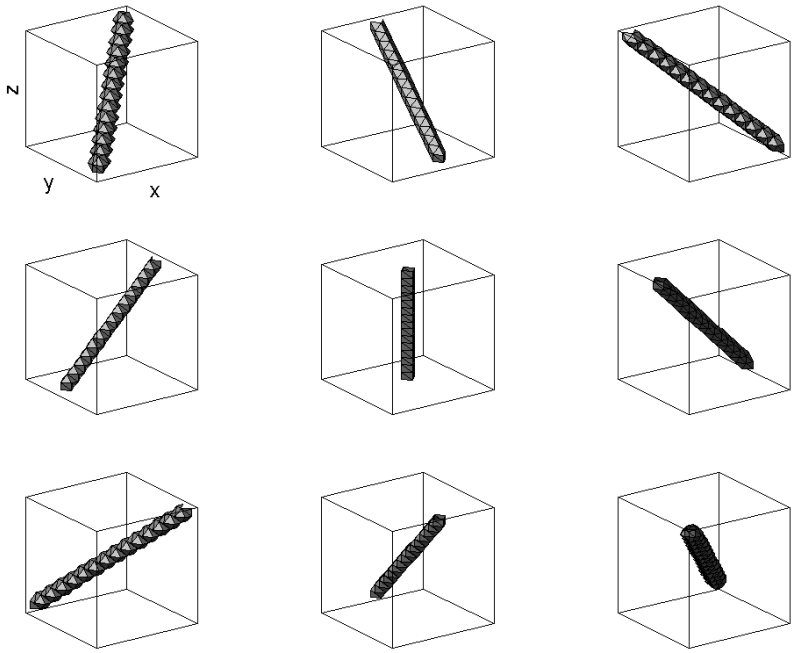

Figure 15. Representers of several X-ray coefficients.

It is also worth considering what happens if we apply the adjoint to coefficient vectors which are ones in various regions and zeros elsewhere in coefficient space. Intuitively, the result should be a bundle of lines. Depending on the span of the region in slope and intercept, the result might be simply like a thick rod (if only intercepts are varying) or like a dumbbell (if only slopes are varying). To check this, we depict in Figure 16 backprojection of six different region indicators. With a little reflection, we can see that these are geometrically correct.

It is of interest to consider backprojection of more interesting coefficient arrays, such as wavelets with vanishing moments. We have done so and will discuss the results elsewhere.

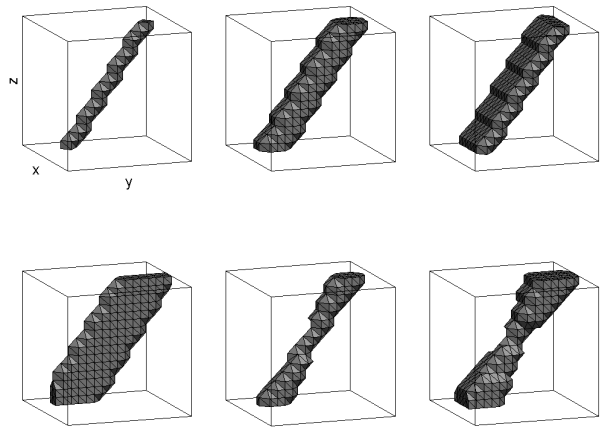

Figure 16. X-ray back-projections of various rectangles in coefficient space. Note that if the rectangle involves intercepts only, the backprojection is rectangular (until cut off by cube boundary). If the rectangle involves slopes, the backprojection is dumbbell-shaped (see lower right)

7.2. Analysis. Now that we have the ability to generate linelike objects in 3-D via backprojection from the X-ray domain, we can conveniently investigate the properties of X-ray analysis.

Consider the example given in Figure 17. A beam is generated by backprojection as in the previous section. It is then analyzed according to the X-ray transform. If the X-ray transform were orthogonal, then we would see perfect concentration of the transform in coefficient space, at precisely the location of the spike used to generate the beam. However, the transform is not orthogonal, and what we see is a concentration — but not perfect concentration — in coefficient space near the location of the true generator.

Also, if the transform were orthogonal, the rearranged sorted coefficients would have a single nonzero coefficient. As the figure shows, the coefficients decay linearly on a semilog plot, indicating power-law decay. The lower right subpanel shows the decay of the wavelet-X-ray coefficients that are computed by applying a four dimensional periodic orthogonal wavelet transform to the X-ray coefficients. As expected, the decay is much faster than the decay of the X-ray coefficients.

8. Application: Detecting Fragments of a Helix

We now sketch briefly an application of beamlets to detecting fragments of a helix buried in noise. We suppose that we observe a cube of noisy 3-D data, and that, possibly, the data contains (buried in noise) a filamentary object. By 'filamentary object' we mean the kind of situation depicted in Figure 18. A series of pixels overlapping a nonstraight curve is highlighted there, and we imagine

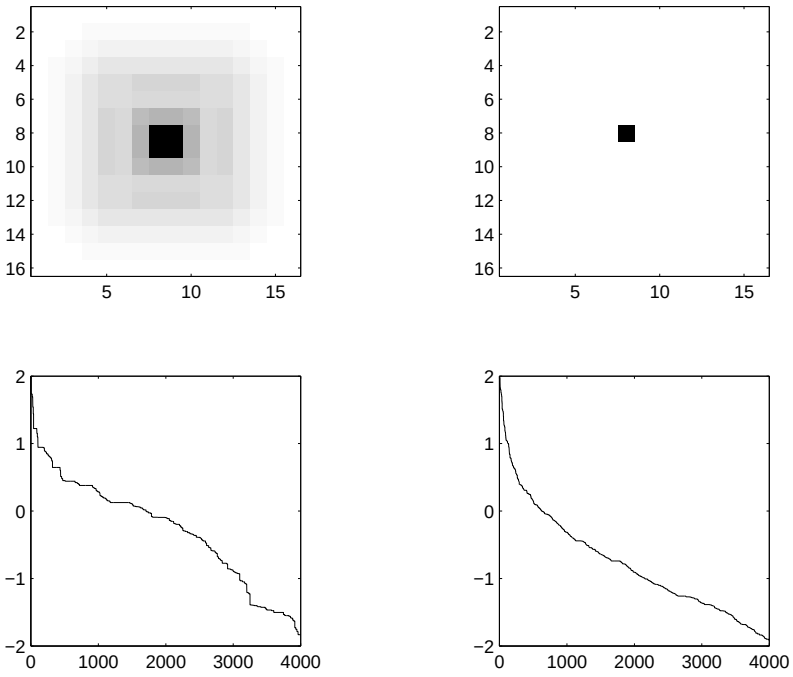

Figure 17. X-Ray analysis of a beam. (a) The X-ray transform sliced in the constant-intercept plane. (b) The X-ray transform sliced in the constant-slope plane. (c) The sizes of sorted X-ray coefficients. (d) The sizes of sorted wavelet-X-ray coefficients.

that, when such an object is 'present' in our data, that a constant multiple of that 3-D template is added to a pure noise data cube.

Figure 18. A noiseless helix.

When this is done, we have a situation that is hard to depict graphically, since one cannot 'see through' such a noisy cube. By this we mean the following: to

visualize such a data cube, it seems that we have just two rendering options. We can view the cube as opaque, render only the surface, and then we certainly will not see what's going on inside the cube. Or we can view the cube as transparent, in which case, when each voxel is assigned a gray value based on the corresponding data value, we see a very uniformly gray object.

Being stymied by the task of 3-D visualization of the noisy cube, we instead display some 2-D slices of the cube; see the rightmost panel of Figure 19. For comparison, we also display the same slices of the noiseless helix. The key point to take away from this figure is that the noise level is so bad that the presence of the helical object would likely not be visible in any slice through the data volume.

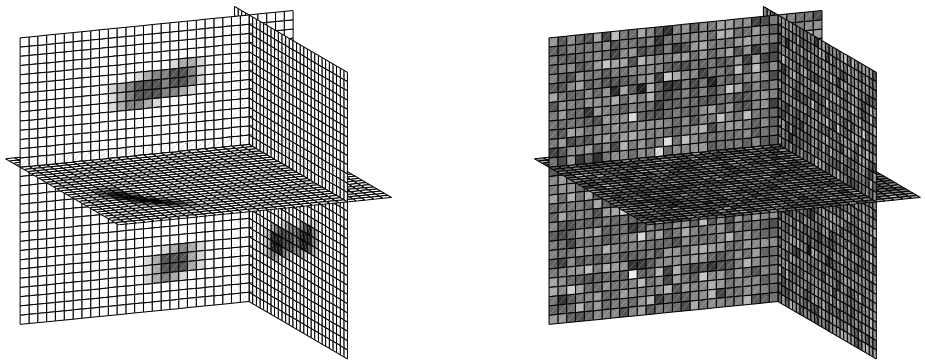

Figure 19. Three orthogonal slices through (a) a noiseless helix; (b) the noisy data volume.

Here is a simple idea for detecting a noisy helix: *beamlet thresholding*. We simply take the beamlet transform, normalize each empirical beamlet coefficient by dividing by the length of the beamlet, and then identify beamlet coefficients (if any) that are unusually large compared to what one would expect if we were in a noise-only situation.

Figure 20 shows the results of applying such a procedure to the noisy data example of Figures 18-19. The extreme right subpanel shows the beamlets that were found to have significant coefficients. The center panel shows the result of backprojecting those significant beamlets; a rough approximation to the filament (far left) has been recovered.

9. Application: A Frame of Linelike Elements

We also briefly sketch an application in using the X-ray transform for data representation. As we have seen in Section 7.1, the backprojection of a delta sequence in X-ray coefficient space is a line-like element. We have so far interpreted this as meaning that the X-ray transform defines an *analysis* of data

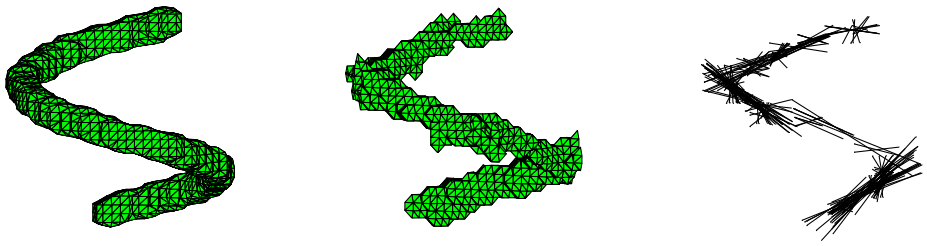

Figure 20. A noiseless helix, a reconstruction from noisy data obtained by backprojecting coefficients exceeding threshold, and a depiction of the beamlets associated to significant coefficients.

via line-like elements. But it may also be interpreted as saying that backprojection from coefficient space defines a synthesis operator, which, for the 'right' coefficient array, can synthesize a volumetric image from linelike elements.

The trick is to find the 'right' coefficient array to synthesize a given desired object. This can be conceptually challenging because the X-ray transform is overdetermining, giving order n^4 coefficients for an order n^3 data cube. Iterative methods for solving large-scale linear systems can be tried, but will probably be ineffective, owing to the large spread in singular values of the X-ray operator.

There is a way to modify the (slant-stack/shearing) X-ray transform to produce something that has reasonably controlled spread of the singular values. This uses the fact, as described in Averbuch et al. [6], that there is an effective preconditioner for the 2-D slant stack operator S (say), such that the preconditioned operator $\tilde{S}$ obeys

$$c_0\|I\|_2 \le \|\tilde{S}I\|_2 \le c_1\|I\|_2.$$

Here $c_1/c_0 < 1.1$. Hence, the transform from 2-d images to their coefficients is almost norm-preserving. In effect, $\tilde{S}$ performs a kind of fractional differentiation of the image before applying S. If, in following the construction of the X-ray transform that was laid out in Section 5.1, we simply replace each invocation of S by $\tilde{S}$. Then effectively, the transform coefficients, grouped together in the families $\mathcal{L}_{xz,n}(s_z, t_z)$ have in each such group, roughly the same norm as the data in the corresponding plane $\mathcal{P}_{xz,n}(s_z, t_z)$, say of the data cube. For each fixed slope s_z, the family of planes $\mathcal{P}_{xz,n}(s_z, t_z)$ with different intercepts t_z, fill out the whole data cube, and so the norms of all these planes, combined together by a sum of squares, gives the squared norm of the whole data cube. It follows that the transform of a volumetric image $I(x, y, z)$ should yield a coefficient array with ℓ^2 norm roughly proportional to the ℓ^2 norm of the array I.

Definition 3. The preconditioned X-ray transform $\tilde{X}$ is the result of following the prescription for Section 5.1 to build an X-ray transform, only using the preconditioned slant stack rather than the slant stack.

We should note that in the theory of the continuum X-ray transform [51], there is the notion of *X-ray isometry*, which preserves the L^2 norm while mapping from physical space to line space. This can be viewed as applying the X-ray transform to a fractional differentiation of the object f, rendering the whole system an isometry. The preconditioned digital X-ray operator $\tilde{X}$ we have just described is a digital analog, although it does not provide a precise isometry.

Standard facts in linear algebra (e.g. [28; 30]) imply that, because the output norm $\|\tilde{X}I\|_2$ is (roughly) proportional to the input norm $\|I\|_2$, iterative algorithms (relaxation, conjugate gradients, etc.) should be able to efficiently solve equations $\tilde{X}I = y$.

The X-ray transform is highly redundant (as it maps n^3 arrays into $O(n^4)$ arrays). As a way to obtain greater sparsity, one might consider applying an orthogonal wavelet transform to the X-ray coefficients. This will preserve the norm of the coefficients, while it may compress the energy into a few large coefficients. The transform is (naturally) 4-dimensional, but as the display in Figure 17 suggests, our concern is more to compress in the slope variable where the analysis of a beam is spread out, rather than in the intercept variables, where the analysis of a beam is already compressed.

DEFINITION 4. The wavelet-compressed X-ray transform $\widetilde{WX}$ is the result of applying an orthogonal 4-D wavelet transform to the preconditioned X-ray transform.

Label the coefficient indices in the wavelet-compressed X-ray transform domain as $\lambda \in \Lambda$, and let the entries in $\widetilde{WX}$ be labeled $\alpha = (\alpha_\lambda)$; they are the wavelet-compressed preconditioned X-ray coefficients.

It turns out that one can reconstruct the original image I from its coefficients α. As the wavelet transform is norm-preserving, the map $I \mapsto \widetilde{WX}I$ is proportional to an almost norm-preserving transform, and hence one can go back from coefficient space to image space, using iterative linear algebra. Call this generalized inverse (linear) transformation $\widetilde{WX}^\dagger$. Then certainly $I = \widetilde{WX}^\dagger \alpha$.

This can be put in a more interesting form. The result of applying this generalized inverse transform to a delta coefficient sequence $\delta_{\lambda_0}(\lambda)$ spiking at coefficient index λ_0 (say) provides a volumetric object $\phi_{\lambda_0}(v)$. Hence we may write

$$I = \sum_\lambda \alpha_\lambda \phi_\lambda.$$

The object ϕ_λ is a *frame element*, and we have thus defined a *frame of linelike elements* in 3-space. Emmanuel Candès in personal correspondence has called such things *tubelets*, although we are reluctant to settle on that name for now (tubes being flexible rather than straight and rigid).

In [16] a similar construction has been applied in the continuum case: a wavelet tight frame has been applied to the X-ray isometry to form a linelike frame in the continuum $\mathbf{R}^3$.

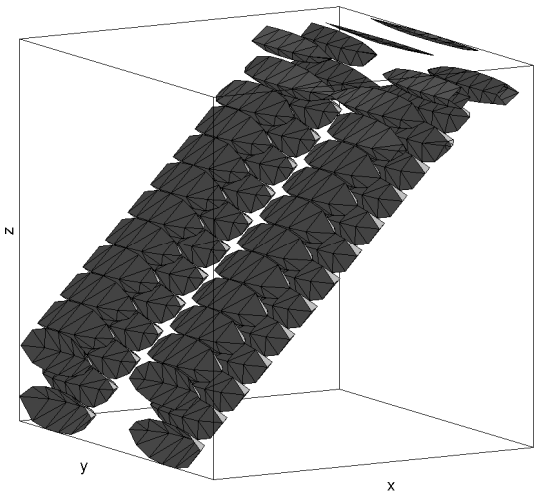

Figure 21. A frame element.

This construction is also reminiscent of the construction of ridgelets for representation of continuous functions in 2-D [14]. Indeed, orthonormal ridgelets can be viewed as the application of orthogonal wavelet transform to the Radon isometry [18]. In [19] a construction paralleling the one suggested here has been carried out for 2-D digital data.

10. Discussion

We finish up with a few loose ends.

10.1. Availability. The figures in this paper can be reproduced by code which is part of the beamlab package. Point your web browser to http://www-stat.stanford.edu/~beamlab to obtain the software. The software has the ability to reproduce all the figures in this paper and has been produced consistent with the philosophy of reproducible research.

10.2. In practice. There are of course many variations on the above schemes, but we have restrained ourselves from discussing them here, even when they are variations we find practically useful, in order to keep things simple. A few examples:

- We find it very useful to work with an alternative vertex-pair dictionary, where the vertices of beamlets are not at corners of boundary voxels for a dyadic cube, but instead at midpoints of boundary faces of boundary voxels.
- We find it useful to work with slight variations of the slant stack defined in [6], where the angular spacing of lines is chosen differently than in that paper.

Rather than burden the reader with such details, we suggest merely that the interested reader study the released software.

10.3. Beamlet algorithms. As mentioned in the introduction, in this paper we have not been able to describe the use of the graph structure of the beamlets in which two beamlets are connected in the graph *if and only if* they have an endpoint in common. In all the examples above, each beamlet is treated independently of other beamlets. As we showed earlier, every smooth curve can be efficiently approximated by relatively few beamlets in a connected chain. In order to take advantage of this fact we must use some mechanism for examining different beamlet chains. The graph structure affords us such a mechanism.

This structure can be useful because there are some low complexity, network-flow based procedures [43; 27] that allow one to optimize over all paths through a graph. Such paths in the beamlet graph correspond to connected chains of beamlets. When applied in the multiscale graph provided by 2-D beamlets, these algorithms were found in [21] to have interesting applications in detecting filaments and segmenting data in 2-D. One expects that the same ideas will prove useful in 3-D.

10.4. Connections with particle physics. In a series of interesting papers spanning both 2-D and 3-D applications, David Horn and collaborators Halina Abramovicz and Gideon Dror have found several ways to deploy line-based systems in data analysis and detector construction[4; 5; 22]. Most relevant to our work here is the paper [22] which describes a linelike system of feature detectors for analysis of data from 3-D particle physics detectors. Professor Horn has pointed out to us, and we agree, that such methods are very powerful in the right settings, and that the main thing holding back widespread deployment of such methods is the immense size of the number of lines needed to give a comprehensive analysis of 3-D data.

10.5. Connections with tomography and medical imaging. The field of medical imaging is rapidly developing these days, and particularly in the last few years, 3-D tomography has become a 'hot topic', with several major conferences and workshops. What is the connection of this work to ongoing work in medical imaging?

Obviously, the X-ray transform, as we have defined it, is closely connected to problems of medical imaging, which certainly obtain line integrals in 3-space and aim to use these to reconstruct the object of interest.

However, the layout of our X-ray transform is (seemingly) rather different than current medical scanners. Such scanners are designed according to physical and economic constraints which place various constraints on the line integrals which can be observed by the system. In contrast, we have only computational constraints and we seek to represent a very wide range of line integrals in our approach. For example, in an X-ray system, a source is located at a fixed point, and can send out beams in a cone, and the line integrals can be measured by a receiving device (film or other) on a planar surface. One obtains many line integrals, but they all have one endpoint in common. In a PET system, events

in the specimen generate are detected by pairs of detectors collinear with the event. One obtains, by summing detector-pair counts over time, an estimated line integral. The collection of integrals is limited by the geometry of the detector arrays.

Essentially, in the vertex-pairs transform, we contemplate a situation that would be analogous, in PET tomography, to having cubical room, with arrays of detectors lining the walls, floor, and ceiling, and with all pairs of detectors corresponding to lines which can be observed by the system. In (physical) X-ray tomography, our notion of X-ray transform would correspond to a system where there is a 'source wall' and the rest of the surfaces were 'receivers', with the specimen or patients being studied oriented successively standing, prone, facing and in profile to the 'source wall'. The (omnidirectional) X-ray source would be located for a sequence of exposures at each point of an array on the source wall (say).

Neither situation is quite what medical imaging experts mean when they say 3-D tomography. For the last ten years or so, there has been a considerable body of work on so called cone-beam reconstruction in 3-D physical X-ray tomography; see [47; 35]. In an example of such a setting [47], a source is located at a fixed point, the specimen is mounted on a turntable in front of a screen, and an exposure is made by generating radiation, which travels through the specimen and the line integral is recorded by a rectangular array at the the screen. This is repeated for each orientation of the turntable. This would be the equivalent of observing the X-ray transform only for those lines which originate on a specific circle in the $z = 0$ plane, and is considerably less coverage than what we envisage.

In PET imaging there are now so-called 'fully 3-D scanners', such as the CTI ECAT EXACT HR+ described in [46]. This scanner comprises 32 circular detector rings with 288 detectors each, allowing for a total of 77×10^6 lines. While this is starting to exhibit some of the features of our system, with very large numbers of beams, the detectors are only sensitive to lines occurring within a cone of opening less than 30 degrees. The closest 3-D imaging device to our setting appears to be the fully 3-D PET system described in [37; 38; 39] where two parallel planar detector arrays provide the ability to gather data on all pairs of lines joining a point in one detector plane to a point in the other plane. In [38] a mathematical analysis of this system has suggested the relevance of the linogram (known as slant stack throughout our article) to the fully 3-D problem, without explicitly defining the algorithm suggested here. Without doubt, ongoing developments in 3-D PET can be expected to exhibit many similarities to the work in this paper, although it will be couched in a different language and aimed at different purposes.

Another set of applications in medical imaging, to interactive navigation of 3-D data, is described in [10], based on supporting tools [9; 11; 55] which are reminiscent of the two-scale recursive algorithm for the beamlet transform.

10.6. Visibility We conclude with a more speculative connection. Suppose we have 3-D voxel data which are binary, with a '1' indicating occupied and a '0' indicating unoccupied. Then a beam which hits only '0' voxels is 'clear', whereas a beam which hits some '1' voxels is 'occluded'. Question: can we rapidly tell whether a beam is 'clear' or 'occluded', for a more or less random beam?

The question seems to call for rapid calculation of line integrals along every possible line segment. Obviously, if we proceed in the 'obvious' way, the algorithmic cost of answering a such a query is order n, since there are line segments containing order n voxels.

Note that, if we precompute the beamlet transform, we can approximately *answer any query about the clarity of a beam* in order $O(\log(n))$ operations. Indeed the beam can written as a chain of beamlets, and we merely have to examine all those beamlet coefficients checking that they are all zero. There are only $O(\log(n))$ coefficients to check, from Theorem 1 above.

We can also rapidly determine *the maximum distance we can go along a ray before becoming occluded*. That is, suppose we are at a given point and might want to travel in a fixed direction. How far can we go before hitting something?

To answer this, consider the the segment starting at our fixed point and heading in the given direction until it reaches the boundary of the data cube — we obviously wouldn't want to go out of the data cube, because we don't have information about what lies there. Take the segment and decompose into beamlets. Now check that all the beamlets are 'clear', i.e. have beamlet coefficients zero. If any are not clear, go to the occluded beamlet closest to the origin, and divide it into its (at most four) children at the next level. If any are not clear, go to the occluded beamlet closest to the origin, and, once again, divide it into its (at most four) children at the next level. Continuing in this way, we soon reach the finest level, and determine the closest occlusion along that beam. The algorithm takes $O(\log(n))$ operations, assuming the beamlet transform has been precomputed.

This allows for rapid computation of what might be called safety graphs, where for each possible heading one might consider taking from a given point, one obtains the distance one can go without collision. The cost is proportional to $\#\text{headings} \times \log(n)$, which seems to be quite reasonable.

Traditional visibility analysis [23] assumes far more about the occluding objects (e.g. polyhedral structure); perhaps our approach would be more useful when occlusion is very complicated and arises in natural systems subject to direct voxelwise observation.

Acknowledgments

Thanks to Amir Averbuch, Achi Brandt, Emmanuel Candès, Raphy Coifman, David Horn, Peter Jones, Xiaoming Huo, Boaz Shaanan, Jean-Luc Starck, Arne Stoschek and Leonid Yaroslavsky for helpful comments, preprints, and references. Donoho would like to thank the Sackler Institute of Tel Aviv University, and both

authors would like to thank the Mathematics and Computer Science departments of Tel Aviv University, for their hospitality during the pursuit of this research.

References

[1] http://www.isye.gatech.edu/~xiaoming/beamlab.

[2] http://www-stat.stanford.edu/~beamlab, http://www.beamlab.org.

[3] http://www-stat.stanford.edu/~wavelab.

[4] H. Abramowicz, D. Horn, U. Naftali, and C. Sahar-Pikielny, "An orientation selective neural network and its application to cosmic muon identification", *Nucl. Instr. Meth. Phys. Res.* A378 (1996), 305–311.

[5] H. Abramowicz, D. Horn, U. Naftali, and C. Sahar-Pikielny, "An orientation selective neural network for pattern identification in particle detectors", pp. 925–931 in *Advances in neural information processing systems* **9**, edited by M. C. Mozer, M. J. Jordan and T. Petsche, MIT Press 1997.

[6] A. Averbuch, R. Coifman, D. Donoho, M. Israeli, and Y. Shkolnisky, "Fast Slant Stack: A notion of Radon transform for data in a cartesian grid which is rapidly computible, algebraically exact, geometrically faithful and invertible", to appear in *SIAM J. Sci. Comput.*.

[7] J. R. Bond, L. Kofman and D. Pogosyan, "How filaments of galaxies are woven into the cosmic web", *Nature* **380**:6575 (April 1996), 603–606.

[8] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network flows: theory, algorithms, and applications*, Prentice-Hall, 1993.

[9] M. L. Brady, "A fast discrete approximation algorithm for the Radon transform", *SIAM J. Computing* **27**:1 (February 1998), 107–19.

[10] M. Brady, W. Higgins, K. Ramaswamy and R. Srinivasan, "Interactive navigation inside 3D radiological images", pp. 33–40 in *Proc. Biomedical Visualization '95*, Atlanta, GA, IEEE Comp. Sci Press, Loas Alamitos CA, 1995.

[11] M. Brady and W. Yong, "Fast parallel discrete approximation algorithms for the Radon transform", pp. 91–99 in *Proc. 4th ACM Symp. Parallel Algorithms and Architectures*, ACM, New York, 1992.

[12] A. Brandt and J. Dym, "Fast calculation of multiple line integrals", *SIAM J. Sci. Comput.* **20**:4 (1999), 1417–1429.

[13] E. Sharon, A. Brandt, and R Basri, "Fast multiscale image segmentation", pp. 70–77 in *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, 2000.

[14] E. Candès and D. Donoho, "Ridgelets: the key to high-dimensional intermittency?", *Phil. Trans. R. Soc. Lond. A.* **357** (1999), 2495–2509.

[15] S. R. Deans, *The Radon transform and some of its applications*, Krieger Publishing, Malabar (FL), 1993.

[16] D. L. Donoho, "Tight frames of k-plane ridgelets and the problem of representing d-dimensional singularities in $\mathbf{R}^n$", *Proc. Nat. Acad. Sci. USA* **96** (1999), 1828–1833.

[17] D. L. Donoho, "Wedgelets: nearly minimax estimation of edges", *Ann. Stat.* **27**:3 (1999), 859–897.

[18] D. L. Donoho, "Orthonormal ridgelets and linear singularities", *Siam J. Math Anal.* **31**:5 (2000), 1062–1099.

[19] D. L. Donoho and Georgina Flesia, "Digital ridgelet transform based on true ridge functions", to appear in *Beyond wavelets*, edited by J. Schmeidler and G. V. Welland, Academic Press, 2002.

[20] D. Donoho and X. Huo, "Beamlet pyramids: a new form of multiresolution analysis, suited for extracting lines, curves, and objects from very noisy image data", in *Proceedings of SPIE*, volume 4119, July 2000.

[21] D. L. Donoho and Xiaoming Huo, "Beamlets and multiscale image analysis", pp. 149–196 in *Multiscale and multiresolution methods*, edited by T. J. Barth and T. F. Chan and R. Haimes, Lecture Notes in Computational Science and Engineering **20**, Springer, 2001.

[22] Gideon Dror, Halina Abramowicz and David Horn, "Vertex identification in high energy physics experiments", pp. 868–874 *Advances in Neural Information Processing Systems* **11**, edited by M. S. Kearns, S. A. Solla, and D. A. Cohn, MIT Press, Cambridge (MA), 1999.

[23] Frédo Durand, "A multidisciplinary survey of visibility: notes of ACM Siggraph Course on Visibility, Problems, Techniques, and Applications", July 2000. http://graphics.lcs.mit.edu/~fredo/PUBLI/surv.pdf.

[24] P. Edholm and G. T. Herman, "Linograms in image reconstruction from projections", *IEEE Trans. Medical Imaging*, **MI-6**:4 (1987), 301–307.

[25] A. Fairall, *Large-scale structures in the universe*, Chichester, West Sussex, 1998.

[26] J. Frank (editor), *Electron tomography, three-dimensional imaging with the transmission electron microscope*, Kluwer/Plenum, 1992.

[27] D. Geiger, A. Gupta, L. A. Costa, and J. Vlontzos, "Dynamic programming for detecting, tracking and matching deformable contours", *IEEE Trans. on Pattern Analysis and Machine Intelligence* **17**:3 (1995), 294–302.

[28] G. Golub and C. van Loan, *Matrix computations*, Johns Hopkins University Press, Baltimore, 1983.

[29] W. A. Götze and H. J. Druckmüller, "A fast digital Radon transform — an efficient means for evaluating the Hough transform", *Pattern Recognition* **28**:12 (1995), 1985–1992.

[30] A. Greenbaum, *Iterative methods for solving linear systems*, SIAM, Philadelphia, 1997.

[31] L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. Tarjan, "Linear time algorithms for visibility and shortest path problems inside triangulated simple polygons", *Algorithmica* **2** (1987), 209–233.

[32] J. L. Hennessy and D. A. Patterson, pp. 373–427 in *Computer architecture: a quantitative approach*, newblock Morgan Kaufmann, San Francisco, 1996.

[33] G. T. Herman, *Geometry of digital spaces*, Birkhäuser, 1998.

[34] G. T. Herman and A. Kuba, *Discrete tomography: foundations, algorithms and applications.* Birkhäuser, 1999.

[35] G. T. Herman and Jayaram K. Udupa, *3D imaging in medicine*, 2nd Edition, CRC Press, 1999.

[36] X. Huo, *Sparse image representation via combined transforms*, PhD thesis, Stanford, August 1999.

[37] C. A. Johnson, J. Seidel, R. E. Carson, W. R. Gandler, A. Sofer , M. V. Green, and M. E. Daube-Witherspoon, "Evaluation of 3D reconstruction algorithms for a small animal PET camera", *IEEE Trans. Nucl. Sci.* 1996.

[38] P. E. Kinahan, D. Brasse, M. Defrise. R. Clackdoyle, C. Comtat, C. Michel and X. Liu, "Fully 3-D iterative reconstruction of planogram data", in *Proc. Sixth International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine*, Asilomar (CA), October 2001.

[39] D. Brasse, P. E. Kinahan, R. Clackdoyle, C. Comtat, M. Defrise and D. W. Townsend, "Fast fully 3D image reconstruction using planograms", paper 15–207 in *Proc. 2000 IEEE Nuclear Science and Medical Imaging Symposium.*

[40] W. Lawton, "A new polar Fourier transform for computer-aided tomography and spotlight synthetic aperture radar", *IEEE Trans. Acoustics Speech Signal Process.* **36**:6 (1988), 931–933.

[41] V. J. Martinez and E. Saar, *Statistics of the galaxy distribution*, Chapman and Hall, 2001.

[42] D. Marr, *Vision : a computational investigation into the human representation and processing of visual information*, W. H. Freeman, San Francisco, 1982.

[43] U. Montanari, "On the optimal detection of curves in noisy pictures", *Comm. ACM* **14**:5 (1971), 335–345.

[44] J. E. Pasciak, "A note on the Fourier algorithm for image reconstruction", Preprint AMD 896, Applied Mathematics Department, Brookhaven National Laboratory (Upton, NY), 1981.

[45] James B. Pawley (editor), *Handbook of biological confocal microscopy*, 2nd edition, Kluwer, 1997.

[46] Jinyi Qi, Richard M. Leahy, Chinghan Hsu, Thomas H. Farquhar, and Simon R. Cherry, "Fully 3D Bayesian image reconstruction for the ECAT EXACT HR+ 1", *IEEE Trans. Nucl. Sci.*, 1997.

[47] A. G. Ramm and A. I. Katsevich, *The Radon transform and local tomography*, CRC Press, Boca Raton (FL), 1996.

[48] B. S. Sathyaprakash, V. Sahni, and S. F. Shandarin, "Emergence of filamentary structure in cosmological gravitational clustering" *Astrophys. J. Lett.* **462**:1 (1996), L5–8.

[49] W. C. Saslaw, *The distribution of the galaxies: gravitational clustering in cosmology*, Cambridge University Press, Cambridge 2000.

[50] Sloan Digital Sky Survey Website: http://www.sdss.org/.

[51] D. C. Solmon, "The X-ray transform", *J. Math. Anal. Appl.* **56** (1976), 61–83.

[52] J.-L. Starck, F. Murtagh, and A. Bijaoui, *Image processing and data analysis*, Cambridge University Press, 1998.

[53] Unattributed, "Imaging system automates volumetric microanalysis", *Vision Systems Design*, Technology Trends Section, June 2001.

[54] M. Unser, P. Thévenaz and L. Yaroslavsky, "Convolution-based interpolation for fast, high-quality rotation of images", *IEEE Trans. on Image Proc.*, **4**:10 (1995), 1371–1381.

[55] T.-K. Wu and M. Brady, "Parallel approximate computation of projection for animated volume-rendered displays", pp. 61–66 in *Proc. 1993 Parallel Rendering Symp.*, 1993.

David L. Donoho
Department of Statistics
Stanford University
Sequoia Hall
Stanford, CA 94305
United States
donoho@stat.stanford.edu

Ofer Levi
Scientific Computing and Computational Mathematics
Stanford University
Gates 2B
Stanford, CA 94305
United States
levi@sccm.stanford.edu