Covariance modeling – practical applications

R. James Purser
SAIC at NOAA/NCEP/EMC

Camp Springs, Maryland

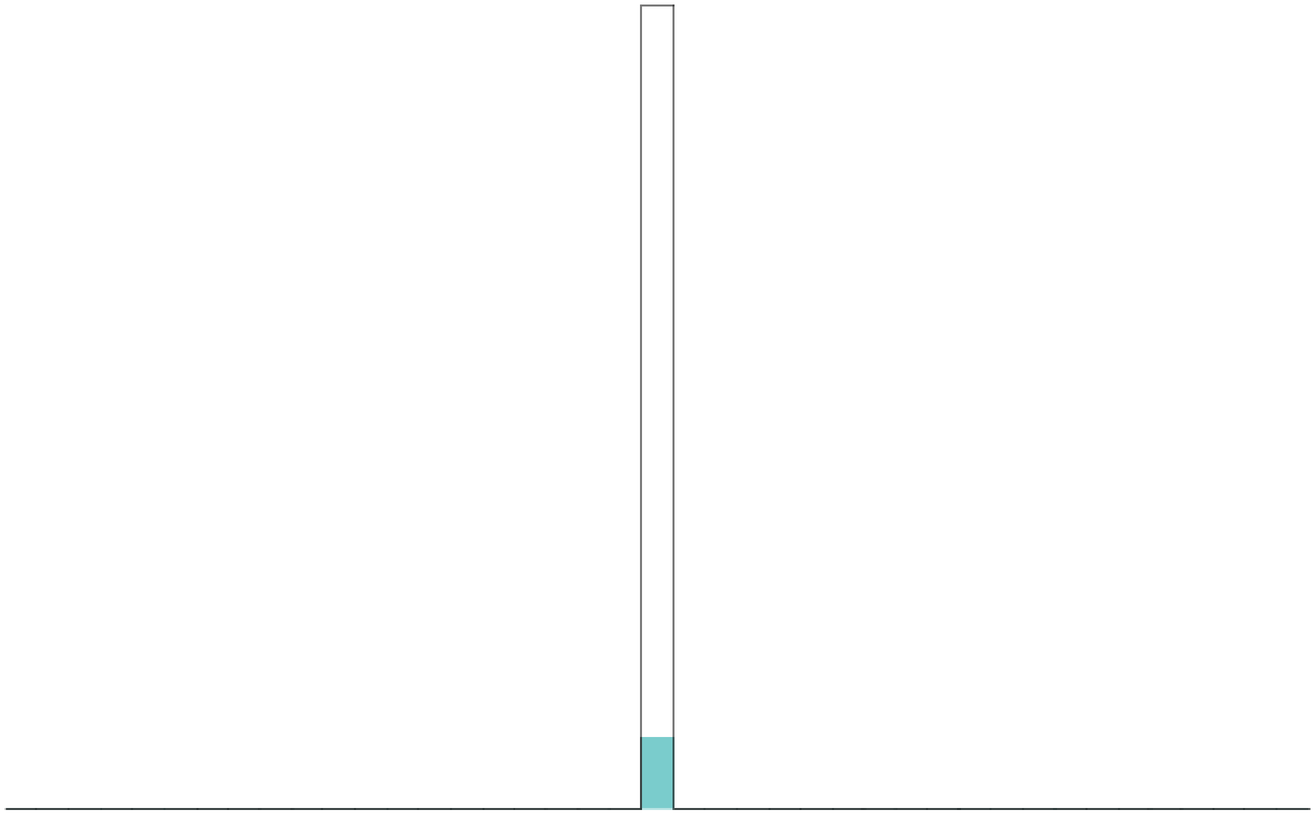
In this talk I want to delve a bit deeper into the practical applications of the recursive filters to the problem of synthesizing plausible and effective error covariances in a numerically highly efficient way.

I will point out a few of the significant challenges and the strategies being adopted to meet them.

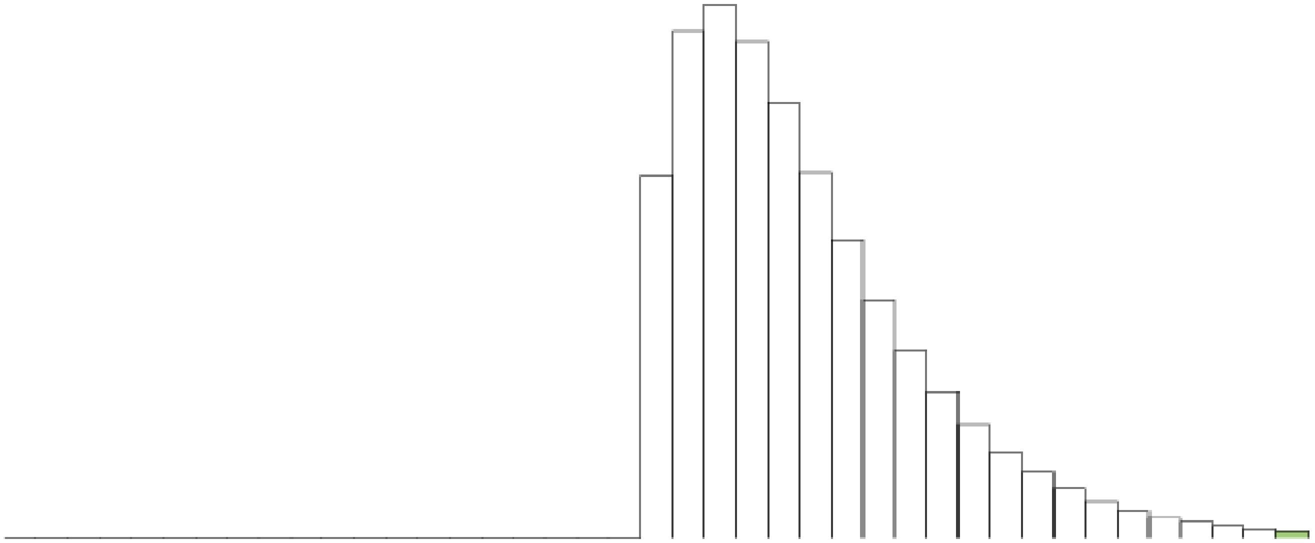
There is an algebraic side to recursive filters which I will merely skim over – the main emphasis will be on the geometrical aspects of the problems (and solutions) that arise. I hope this does not put you off!

Recall how we combined two of the simplest "1st-order" recursive filters, grinding along the grid in one dimension, and in the same direction, into an equivalent single 2nd-order filter.

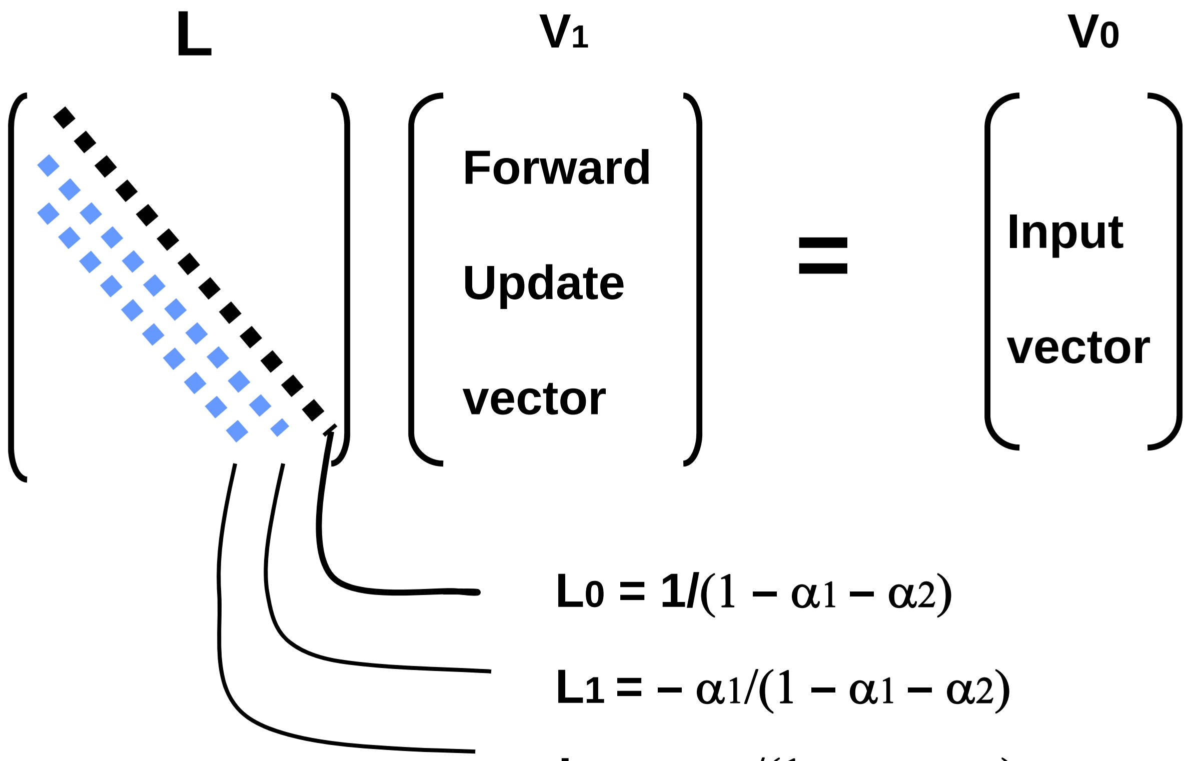
Forwards…



Then backwards….



Let's review how the filters look in matrix/vector terms:



$L_0 = 1/(1 - \alpha_1 - \alpha_2)$

$L_1 = -\alpha_1/(1 - \alpha_1 - \alpha_2)$

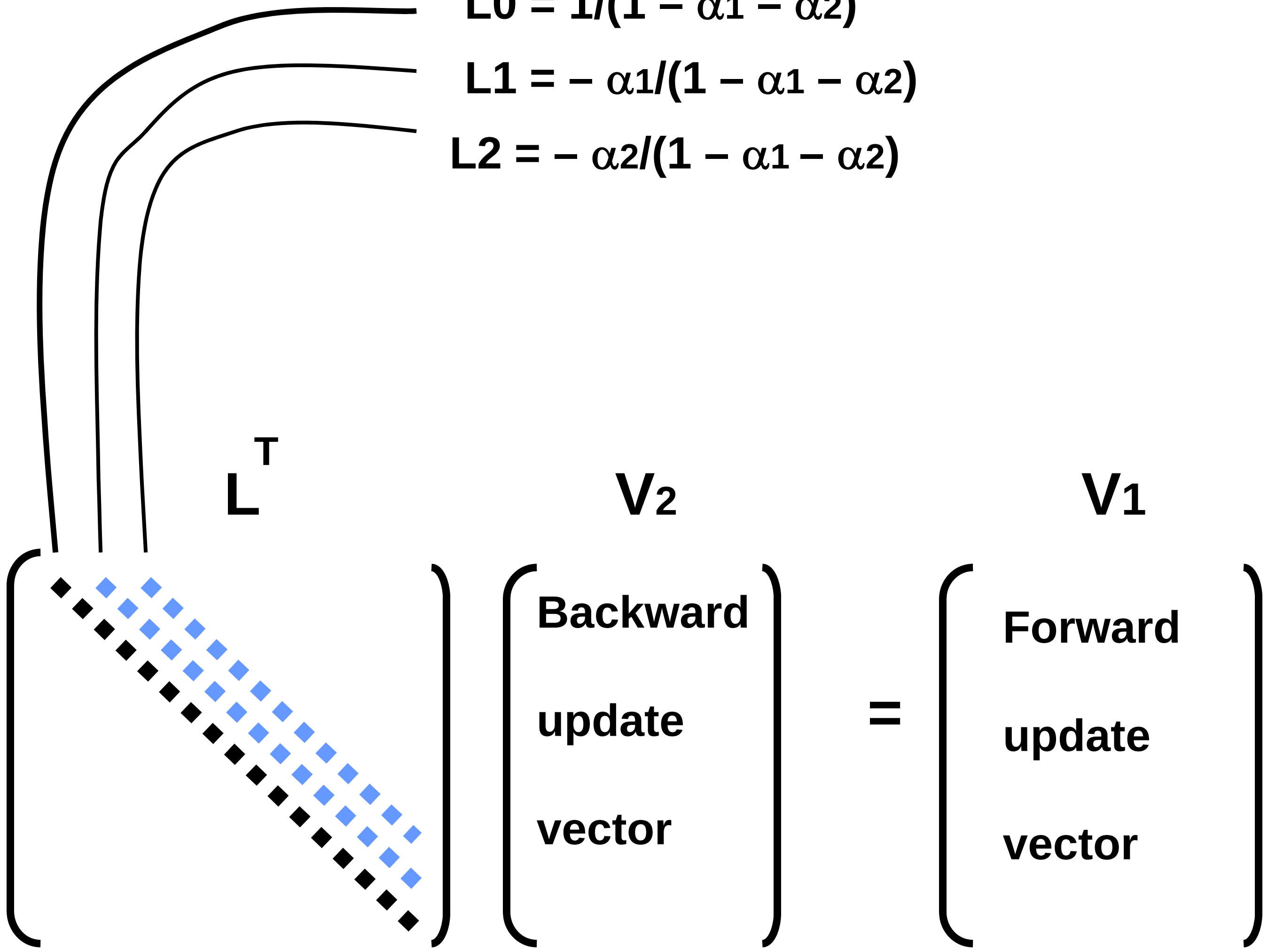
$L_2 = -\alpha_2/(1 - \alpha_1 - \alpha_2)$

The forward sweep

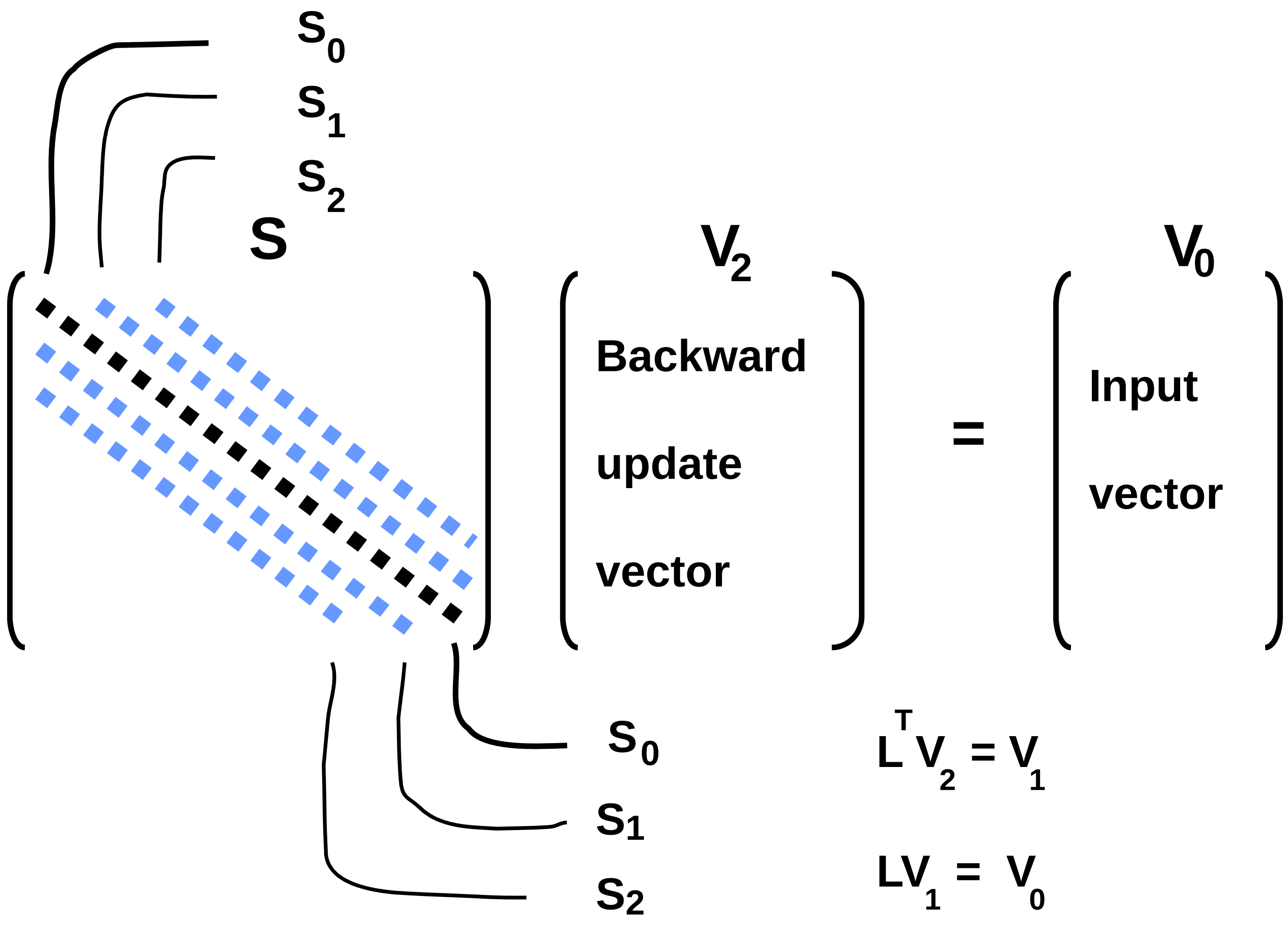
$$L0 = 1/(1 - \alpha_1 - \alpha_2)$$

$$L1 = -\alpha_1/(1 - \alpha_1 - \alpha_2)$$

$$L2 = -\alpha_2/(1 - \alpha_1 - \alpha_2)$$



The backward sweep



$$L^T V_2 = V_1$$

$$L V_1 = V_0$$

$$S V_2 = V_0, \quad \text{where} \quad S = LL^T$$

The one sided filters come from the CHOLESKY factors of S.

How do we make the rows and columns of the INVERSE of S look like a Gaussian?

Answer: Think spectrally!

The Fourier transform of a Gaussian filter is itself a Gaussian, ~ G(k), in wavenumber k.

Suitably scale, $G^{-1}(k) \sim \exp(+k^2/2)$

But $-k^2$ is just the spectral representation of the 2nd-derivative operator!

The second derivative operator has an approximate representation as a tri-diagonal matrix. Higher powers form band matrices.

The solution to the problem of producing a quasi-Gaussian recursive filter is to base its inverse, S, on the finite Taylor expansion, in second-derivative, k^2, of the exponential function, $\exp(k^2/2)$:

$$\sim 1 + k^2/2 + (1/2!)\,(k^2/2)^2 \ldots$$

Lets look at the circularity of the contours of a few 2D combinations of recursive filters:

1st-order

2nd-order

4th order

1st-order applied four times.

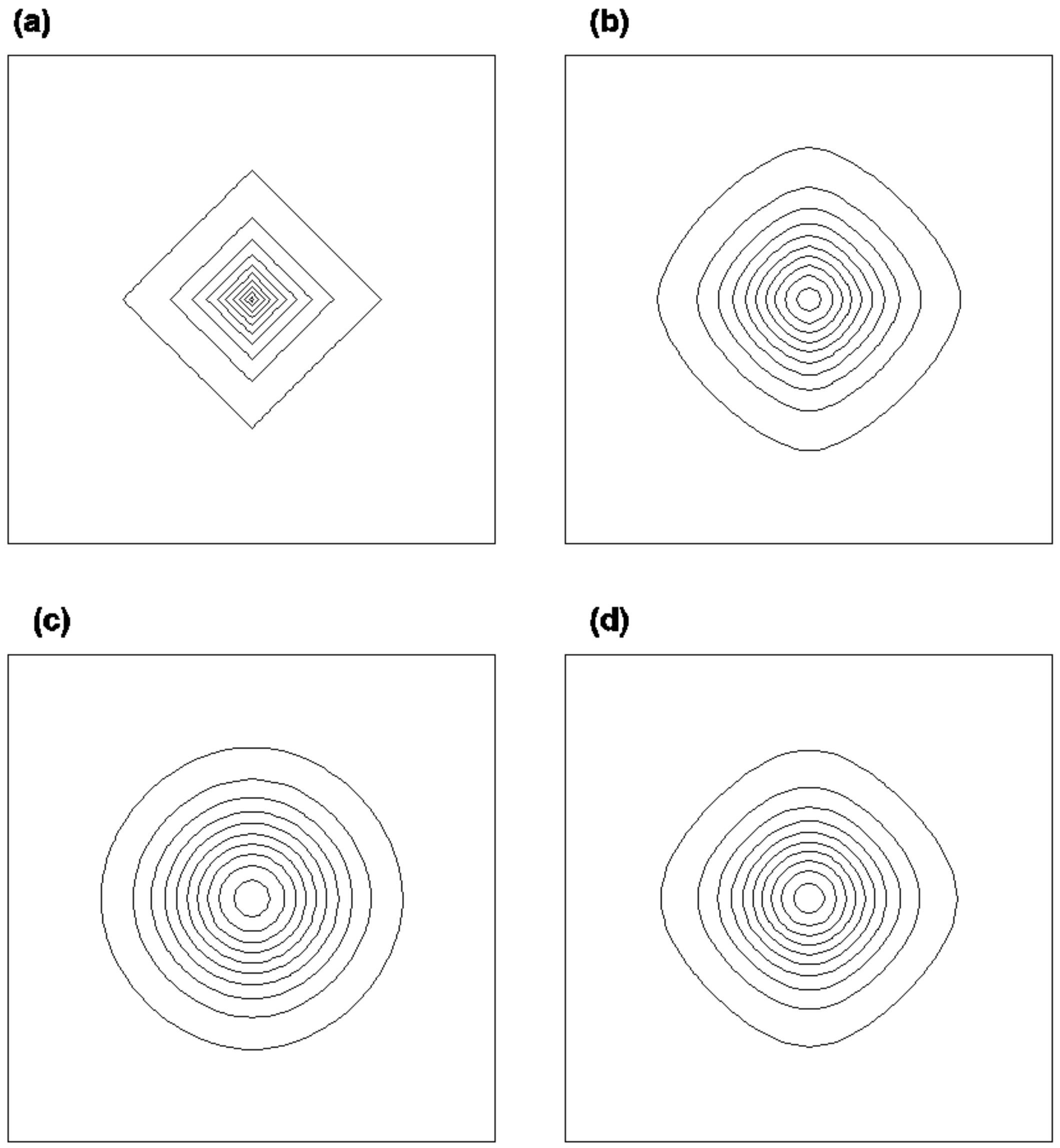
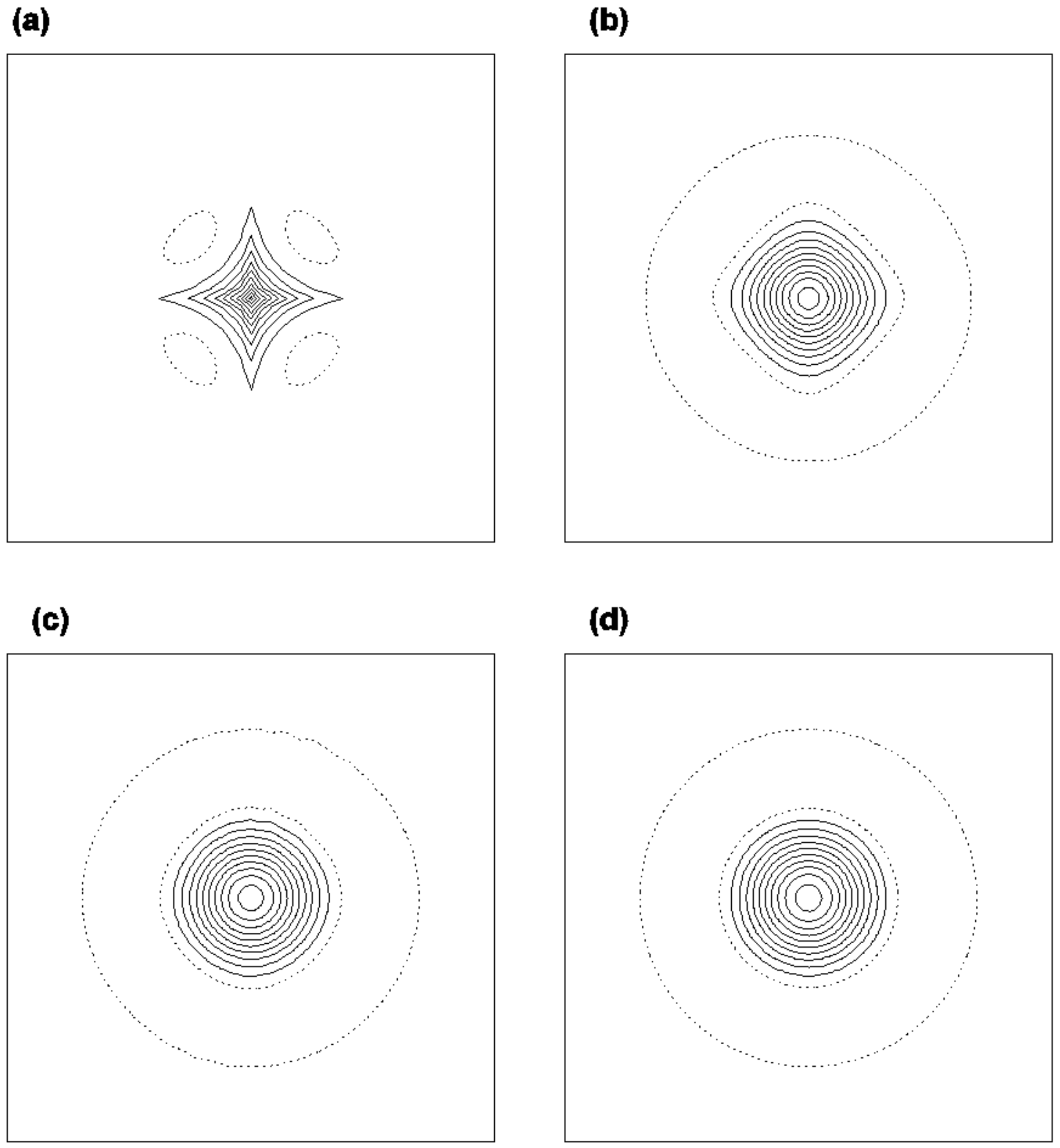


Figure 2. Sequential application of quasi-Gaussian recursive filters of order n in two dimensions. (a) $n = 1$; (b) $n = 2$; (c) $n = 4$; (d) four applications of filters with $n = 1$ with scale parameter adjusted to make the result comparable with the other single-pass filters. Contours are shown at multiples of odd integers.

Negative-Laplacian of filter responses

2nd-order

4th-order



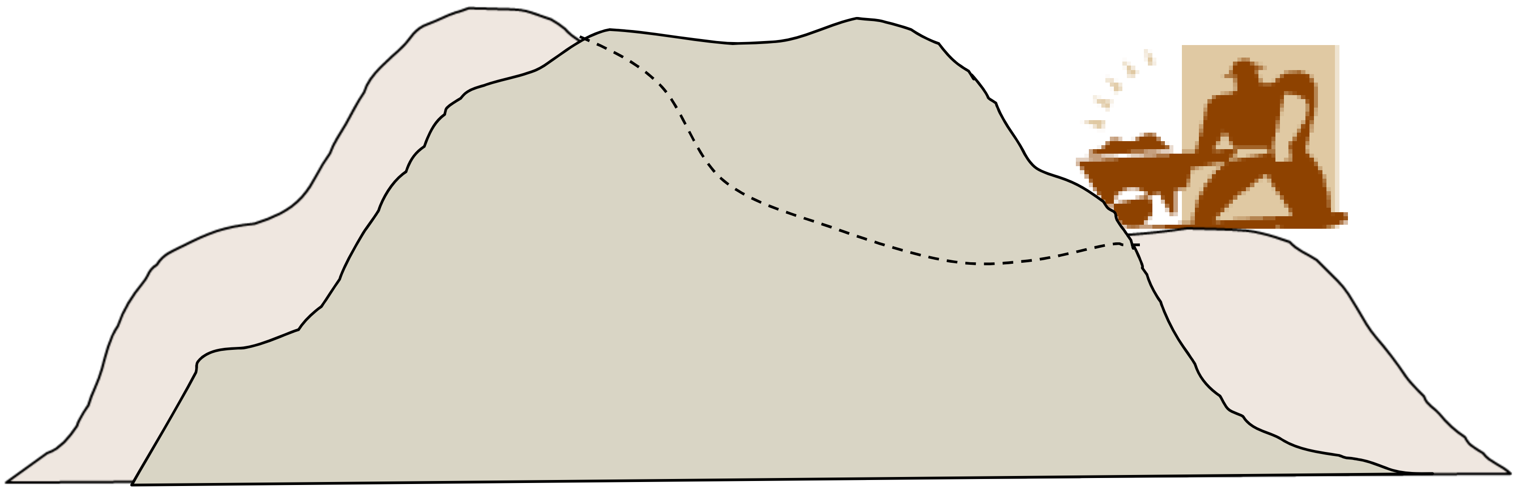
6th-order

True Gaussian

Pictorial displays of the contours of the filter responses are certainly valuable and informative.

But it is also valuable to have an **objective quantitative** measure.

Since we know that the target shape for the 1D filter is the Gaussian, it would be enough just to find an objective measure of the "distance" between the actual 1D filter response profile and the ideal Gaussian profile of the same standardized width.



The Wasserstein “earth-mover’s” metric.

Given two distributions of equal measure, we can define a “distance” or “metric” between them. The earth-mover’s metric, a special case of the more general Wasserstein metric, may be thought of as the minimum number of “wheel-barrow miles” needed to transform a “mountain” of one distribution’s shape into the mountain of the shape of the other distribution.

Table of Wasserstein distance between iterated recursive filter and Gaussian

m = number of iterations of the filter

n = degree of the recursive filter

	n=1	2	3	4	5	6
m = 1	0.14549	0.04366	0.01560	0.00608	0.00250	0.00106
2	0.08160	0.01628	0.00393	0.00104	0.00030	0.00009
3	0.05687	0.00860	0.00158	0.00033	0.00008	0.00004
4	0.04368	0.00534	0.00080	0.00014	0.00005	0.00004
5	0.03547	0.00365	0.00047	0.00008	0.00004	0.00004
6	0.02987	0.00266	0.00030	0.00006	0.00004	0.00004

The Wasserstein metric suggests that the use of high-order filters is a more cost-effective way of approximating the Gaussian ideal than simply applying the first-order filter many times.

When covariance statistics become spatially inhomogeneous, then the bias introduced by high-order filters (which then becomes large) can become a problem. Another problem is that high-order recursive filters, especially those for which the characteristic scale has become much larger than grid scale, are quite ill-conditioned and round-off errors seriously spoil the results with numerical noise.

However, both of these defects of the higher order filters can be partially mitigated by factoring the Taylor-series polynomial into its real linear and quadratic parts (there are no linear factors when the filter degree is even, though). This keeps the bias relatively small when the forward and backward sweeps are alternated, and the ill-conditioning is much improved (though not entirely eliminated at large characteristic scales).

4th-order filters seem a good choice overall.

The 4th-order filters also allow a non-standard factorization which can improve the performance of the filtering code in multidimensional inhomogeneous contexts.

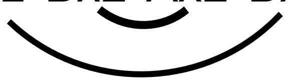
Recall that a 4th Taylor series approximation to the exponential will have a factorization into two quadratic products, each associated with A 2nd-degree recursive filter. Call the forward recursive components, Ax1 and Ax2 and the backward recursive components, Bx1 and Bx2, when they act in the "x" direction, with Ay1, Ay2, By1, By2, the corresponding filters in the "y" direction.

Ax1 is the exact adjoint of Bx1, Ax2 the adjoint of Bx2, etc., regardless of inhomogeneity. The complete self-adjoint x filter, Sx, factors either as Sx = Ax1*Bx1*Ax2*Bx2 or as Sx=Ax2*Bx2*Ax1*Bx1 (order of application being from left factor to right factor). However, while Sx is self-adjoint, the multidimensional combination, Sx*Sy, is not, since Sx*Sy is NOT equal to Sy*Sx.

Strictly, we should use a combination of filters like:
S = Sx*Sy*Sy*Sx (2D) or S = Sx*Sy*Sz*Sz*Sy*Sx (3D)
since self-adjointness of the complete filter, S, is absolutely a must.

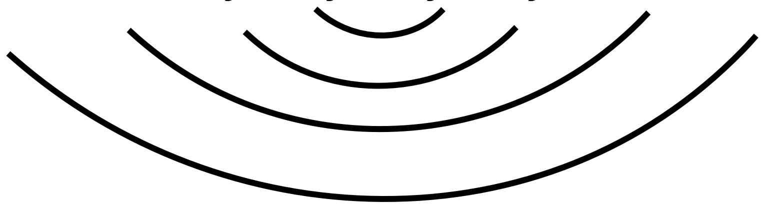
A fortuitous feature of the 4th-degree quasi-Gaussian recursive filter (actually, a property of its polynomial's factors) is that, even in smoothly inhomogeneous environments, the 1D portion, Sx, is APPROXIMATELY factorized by the (EXACTLY) self-adjoint combination:

Sx ~ Ax1*Bx2*Ax2*Bx1.



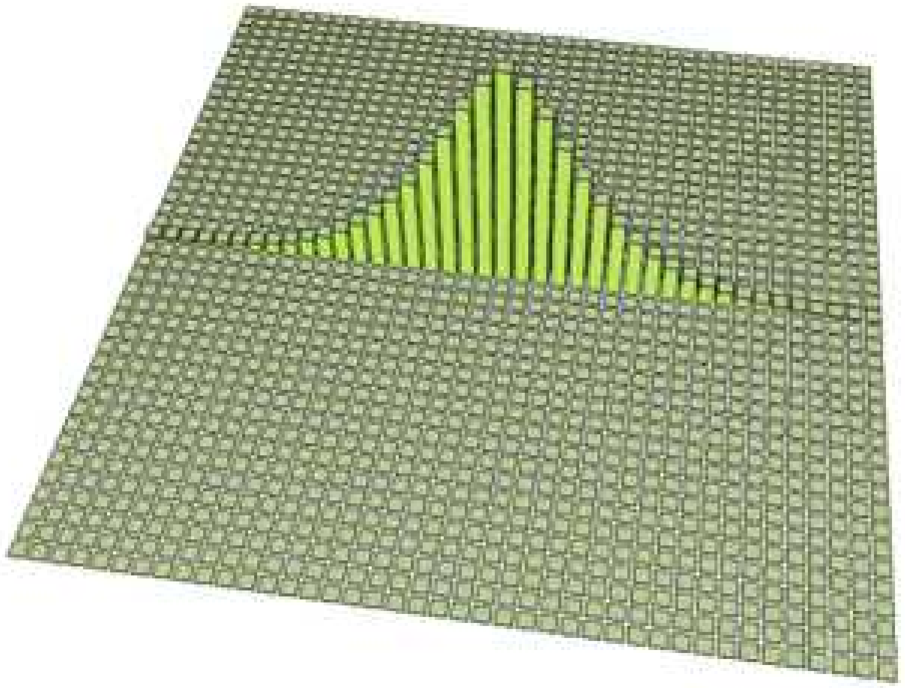
This lets us adopt a shorter (by a factor of two) sequence of factors in the multidimensional case. For example, in 2D:

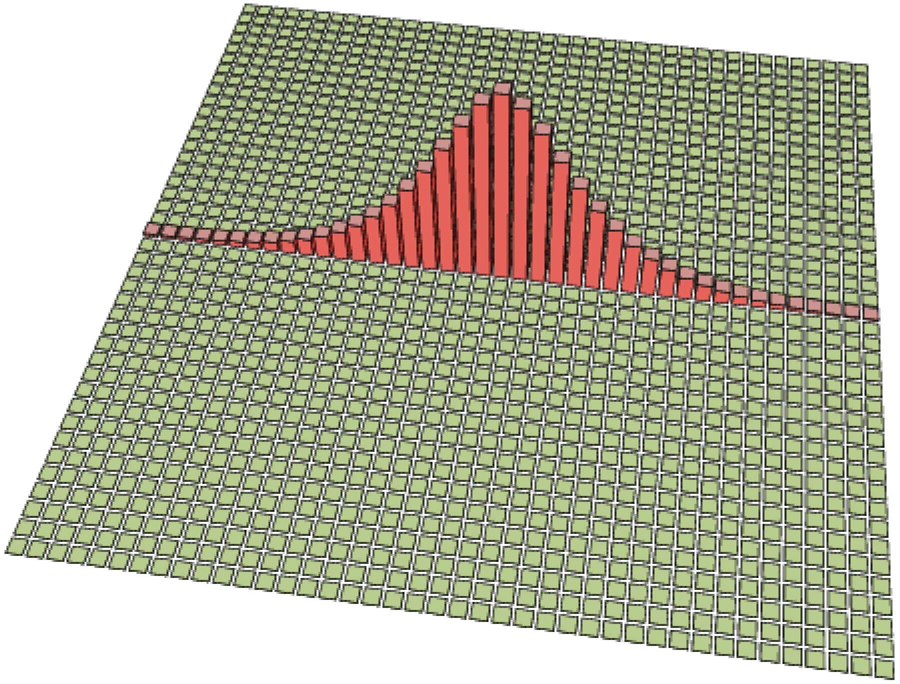
S = Ax1*Bx2*Ay1*By2*Ay2*By1*Ax2*Bx1

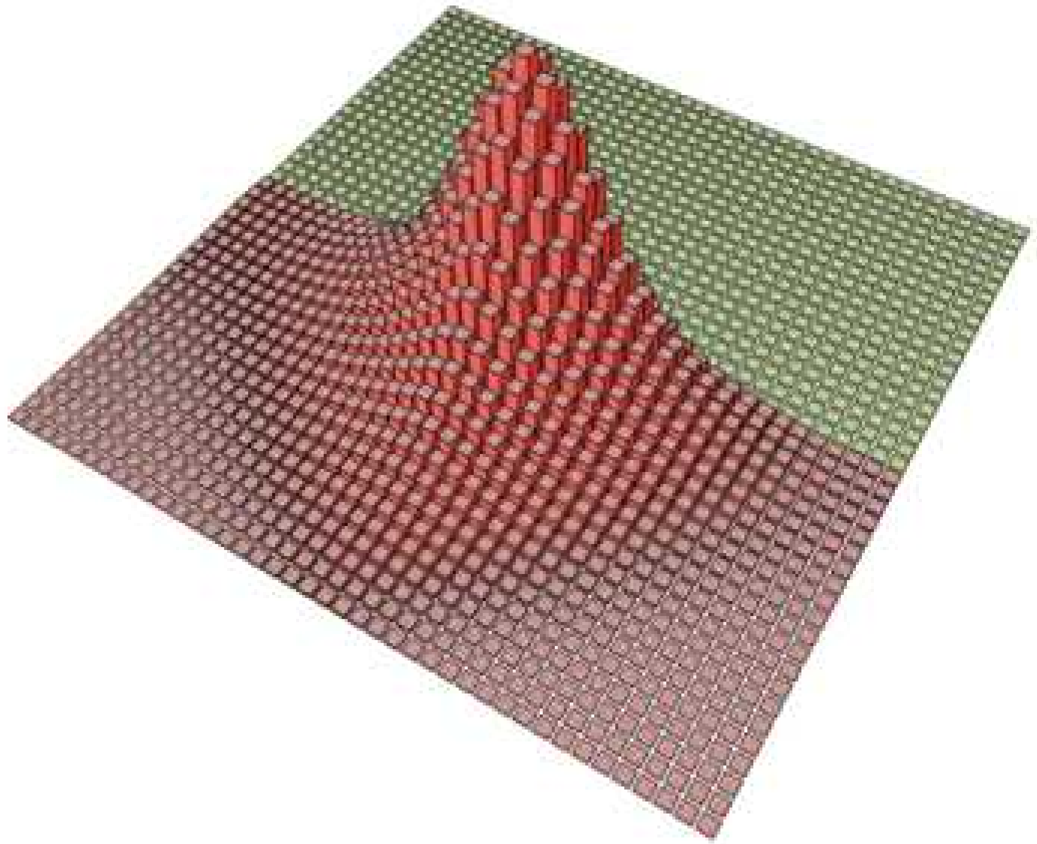


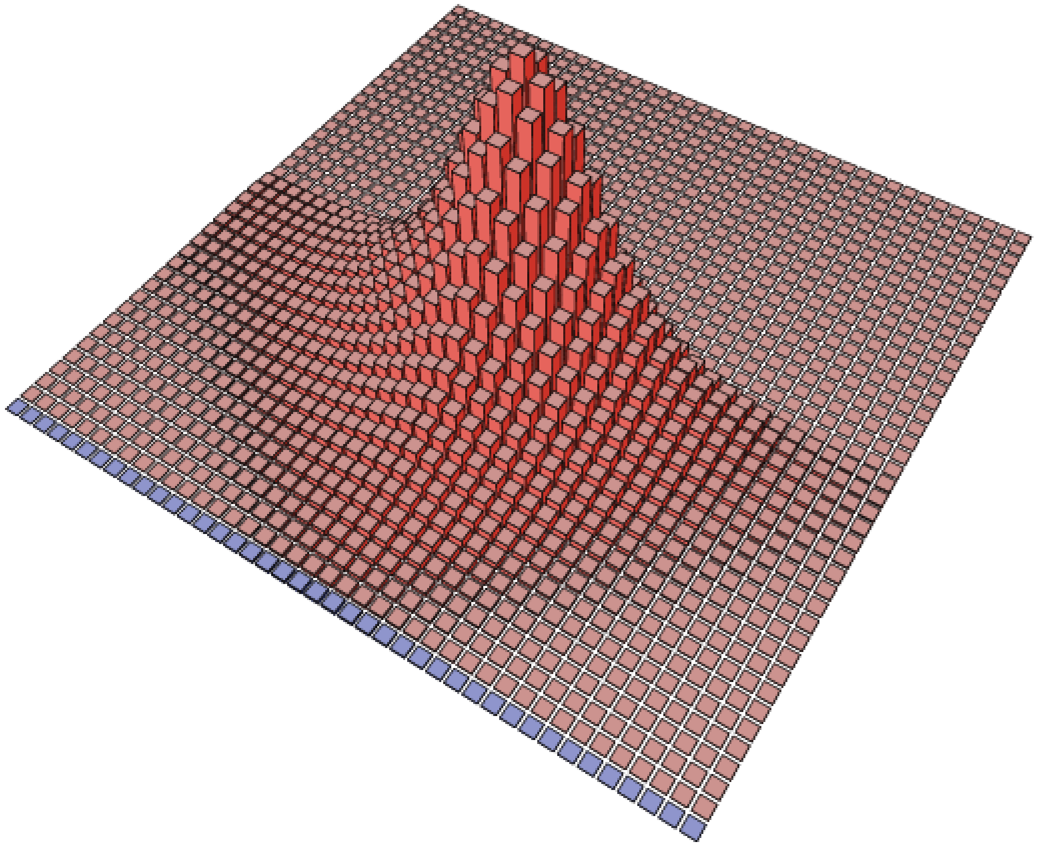
This factorization is being coded in the new RF package.

By why restrict to Sx, Sy and Sz ? There should be no restriction!!

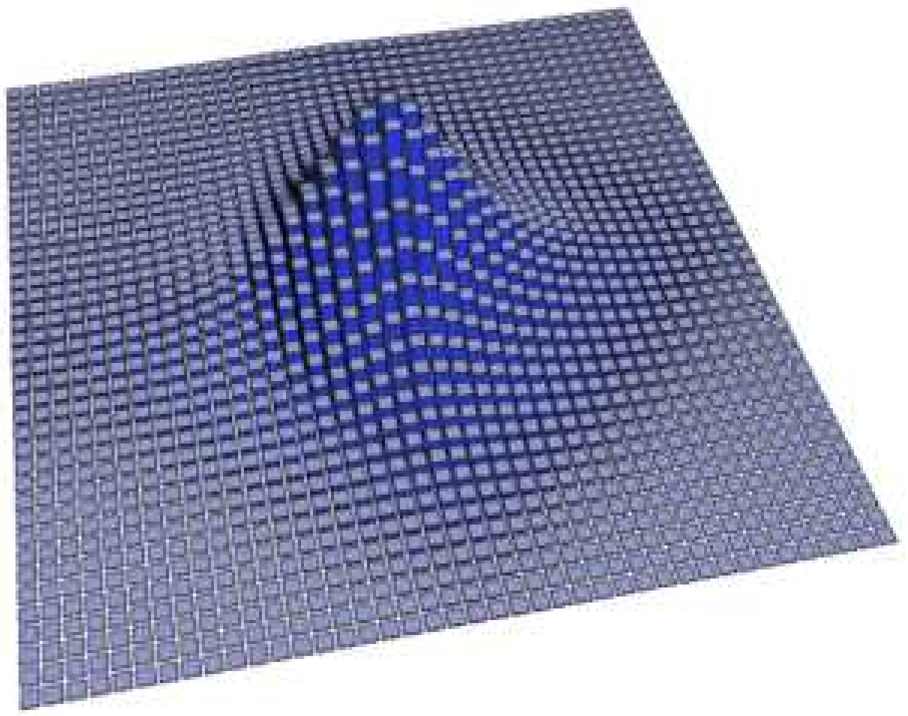




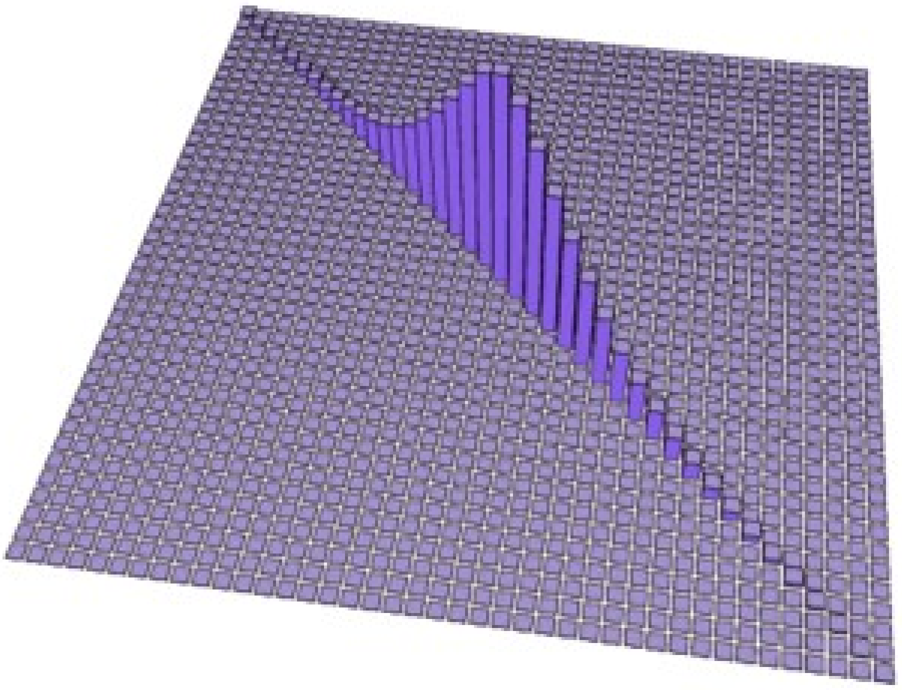




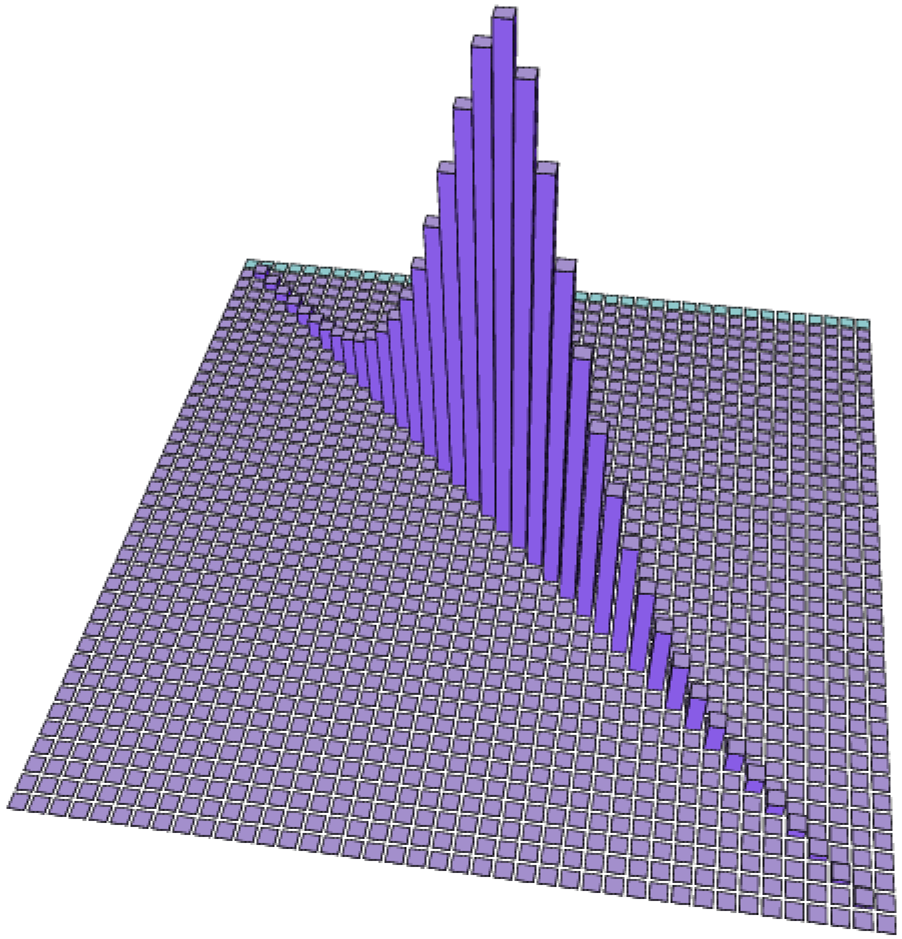
The result is approximately isotropic

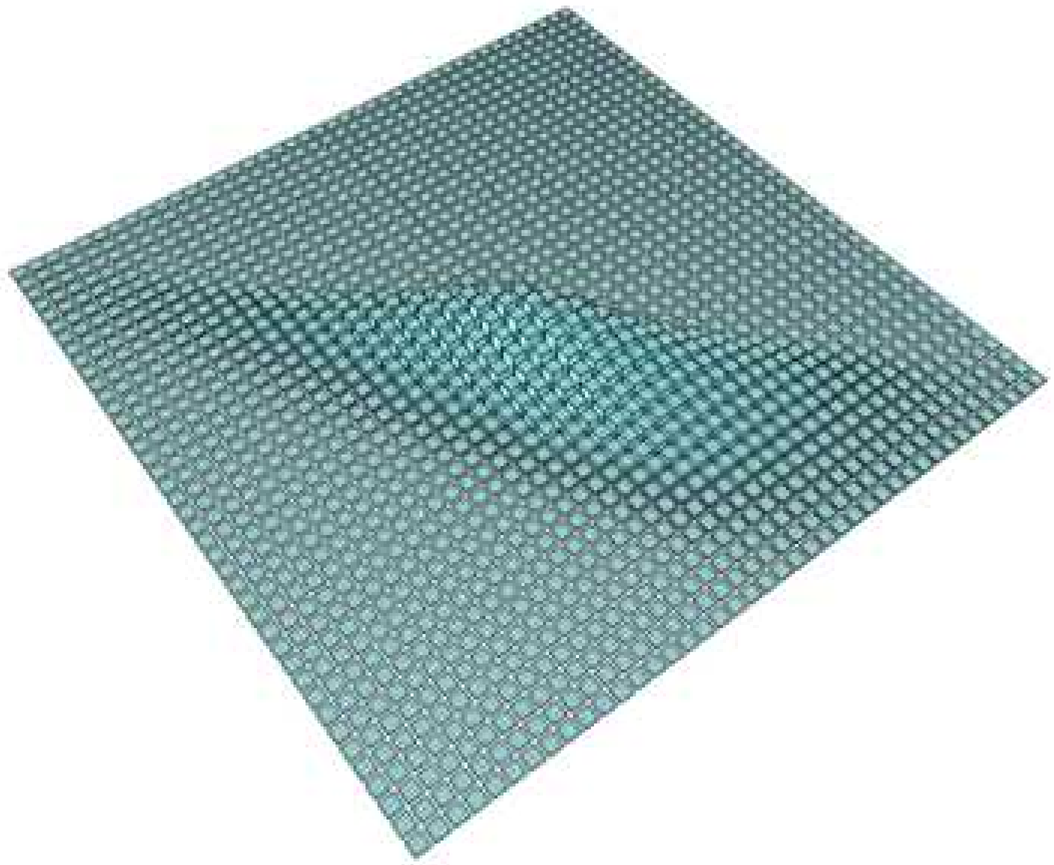


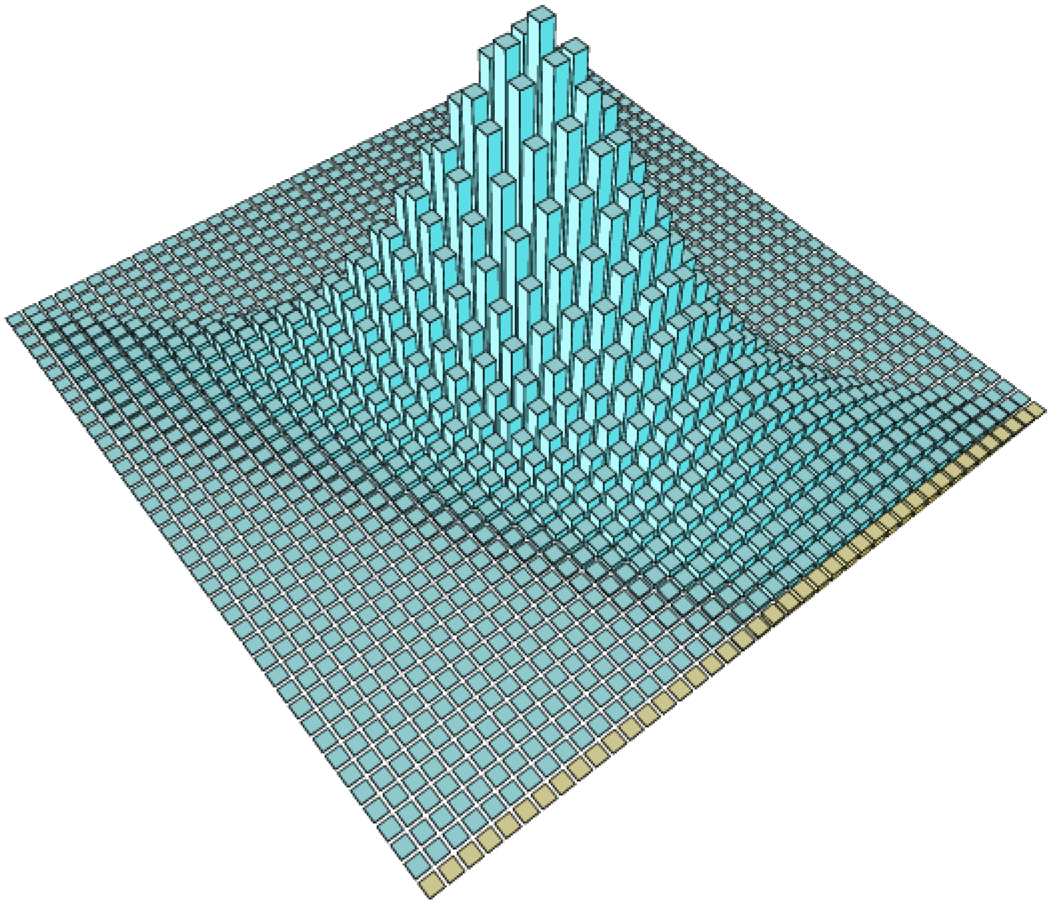
This is also quasi-Gaussian (but degenerate, so far)



In this case, the filtering was along generalized parallel, but “non-Cartesian” lines of the lattice (oriented at 45 degrees).

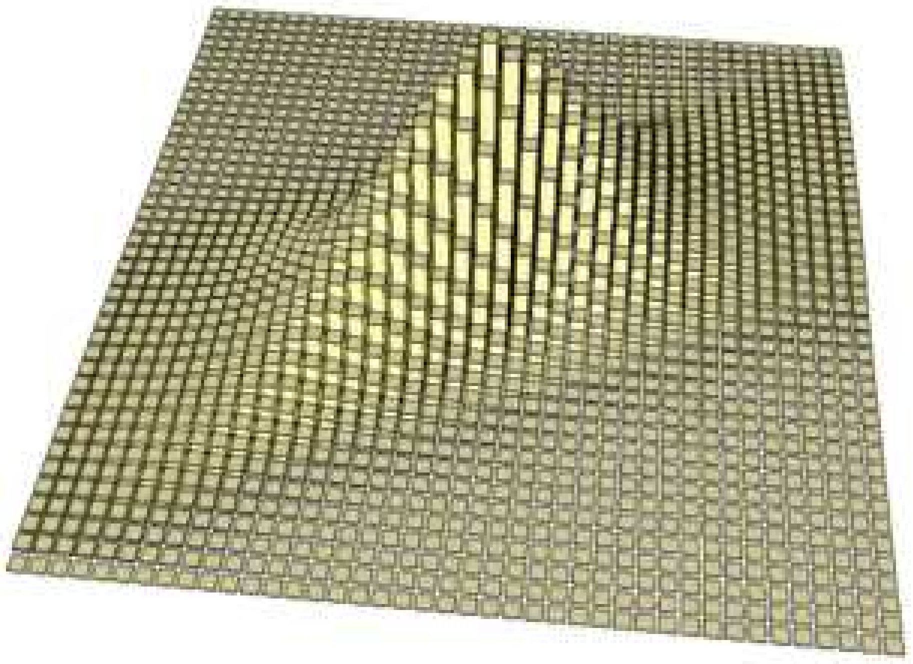






Now we have the end result – quasi-Gaussian, anisotropic, and non-degenerate.

A “STRETCHED COVARIANCE”!



By adopting generalized line filters of the quasi-Gaussian type, we greatly expand the range of available covariance shapes.

Perhaps, by a suitable selection of lines and their associated filtering “weights” (2nd-moments, in the given grid units of each line), it might be possible to synthesize an ANISOTROPIC Gaussian with ANY degree of anisotropy.

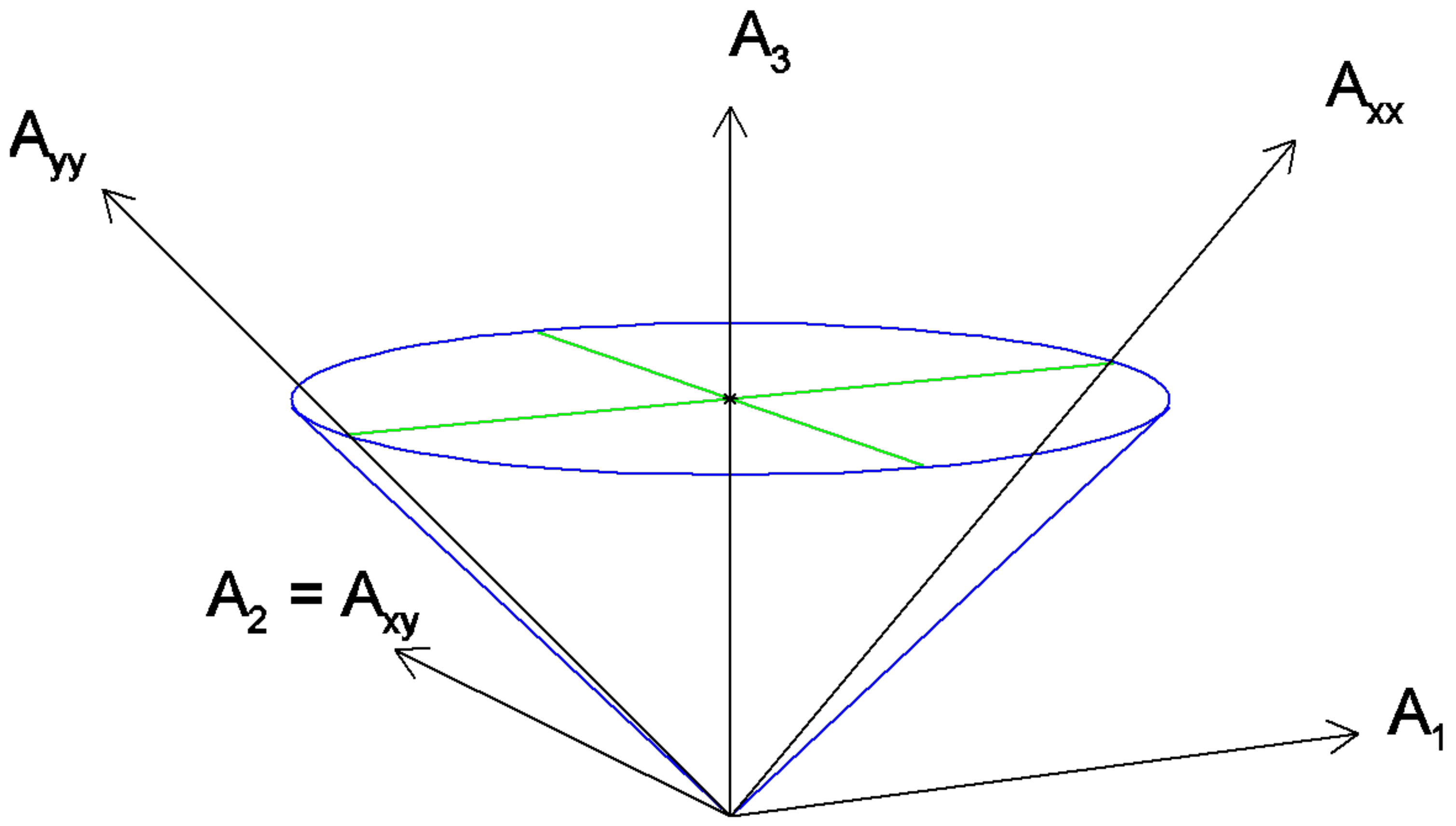
In this case, the (centered and normalized) second moment “spread” of the Gaussian need to be expressed as a tensor. In the given lattice coordinates, it is represented by a symmetric matrix:

$$A = \begin{pmatrix} Axx & Axy \\ Axy & Ayy \end{pmatrix}$$

The “ASPECT” tensor.

Valid (positive semi-definite) aspect tensors lie inside a CONE:

("Aspect cone")



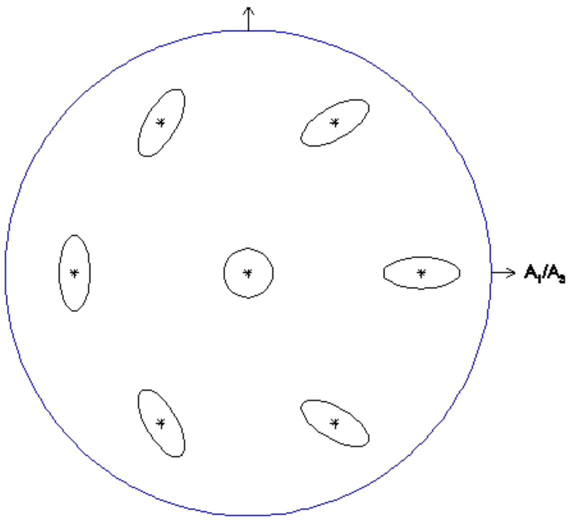
Projected onto the hyperboloid, |A|=1, the aspect tensors exhibit a metric – the distance that measures how much deformation maps one to another. The geometry of this metric is HYPERBOLIC geometry.

The line-filters associated with all possible lines of the lattice each belong to the boundary of the aspect cone.

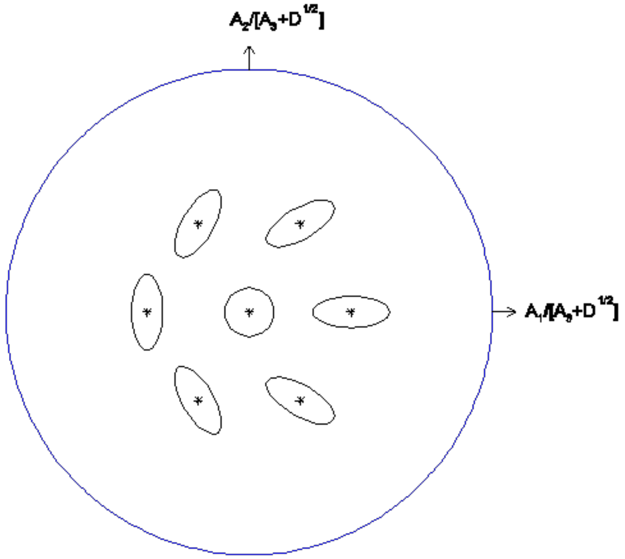
We can see their positions on projective maps of the hyperbolic space. The gnomonic (central) projection is usually referred to as the “Klein” mapping; the stereographic projection is conformal (angle-preserving) and is usually referred to as the “Poincare” mapping (though both were earlier used by Beltrami).

Line-filters do not by themselves produce “proper” covariances that enjoy full-rank aspect tensors. But, in 2D, any sequential pair of such filters will generate a Gaussian with aspect tensor having full rank. The set of aspect tensors that can come from the application of two (parallel families of) line filters of the lattice form a sheet inside the aspect cone whose projection onto the unit-determinant hyperboloid maps to a curve in either the Klein or the Poincare representation. In the Klein map, the curve is a straight chord; in the Poincare map, it is a circular arc that cuts the “limit circle” at right angles.

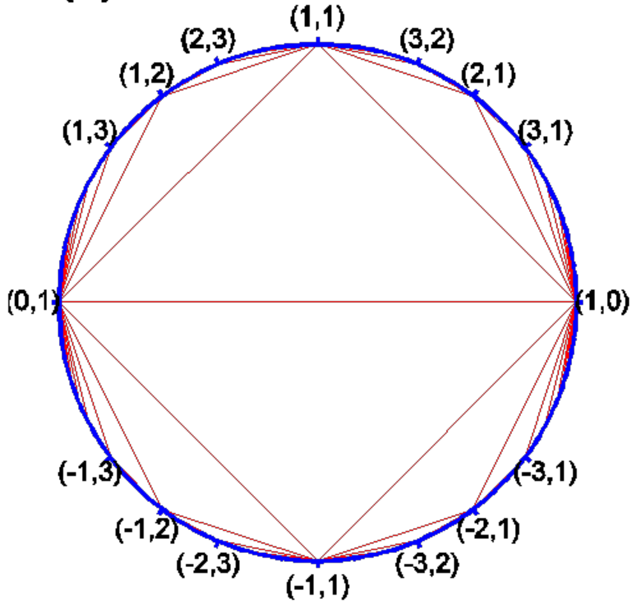
(a)



(b)

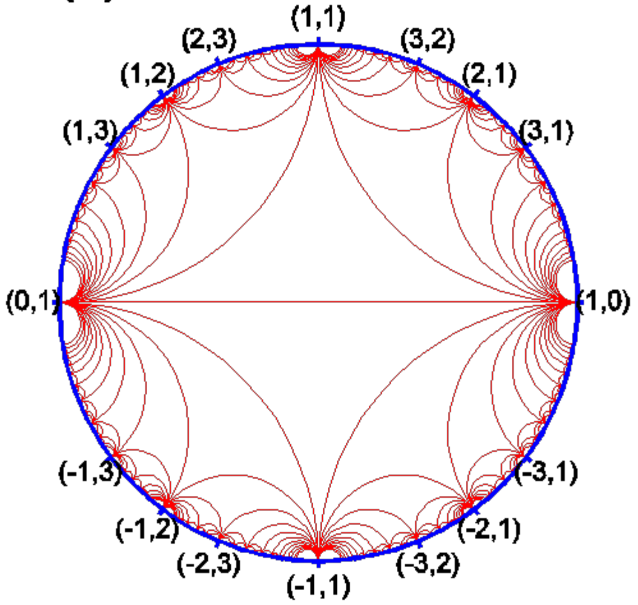


(a) Klein



Klein

(b) Poincaré



Poincaré

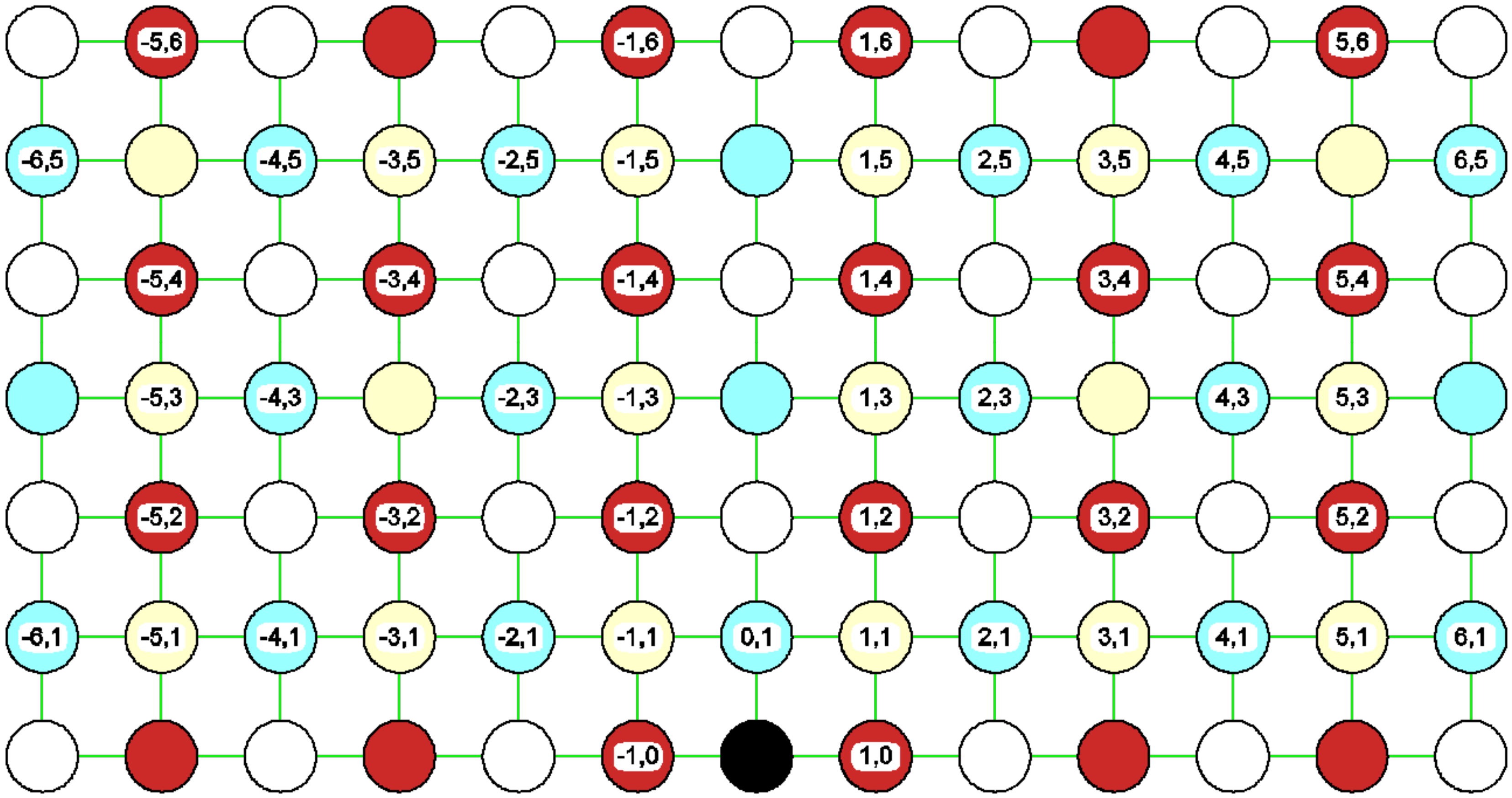
The tiling of the projected space by the regions involving THREE associated line filters ("TRIADS") is complete; therefore ANY 2D aspect tensor can be realized in a Gaussian generated by just three sequentially applied line filters!

The TRIAD ALGORITHM is the means by which a given aspect tensor is resolved into its associated triad of line-directions and filtering "weights". The resolution exploits linearity of 2nd moments under composition by sequential application. (Strictly true only when conditions are spatially homogeneous.)

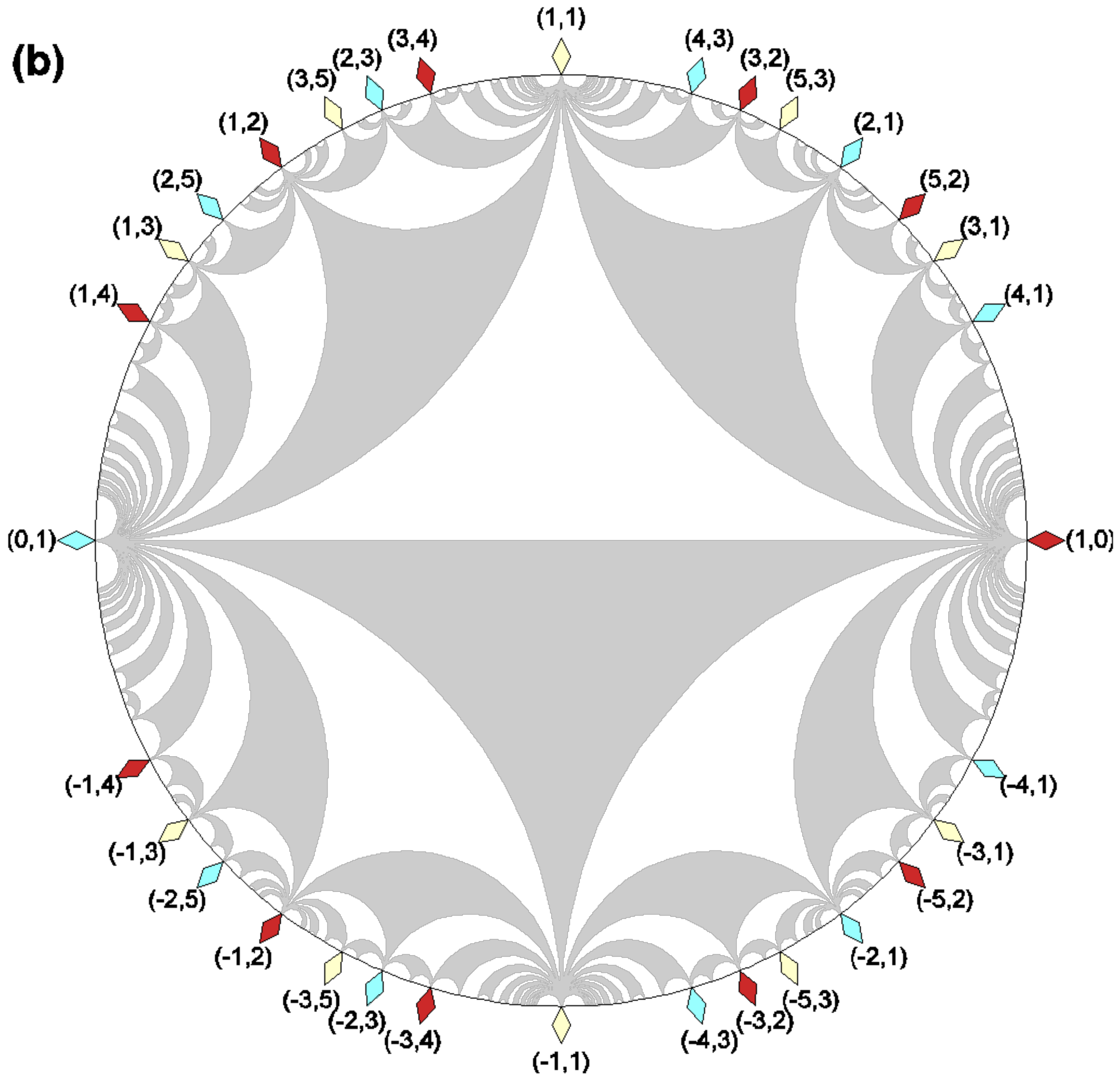
With a palette of three "colors" we can assign a different color to the three directions of a given triad. Having done so, the pattern is extended uniquely to ALL lattice lines by adhering to the condition that all triads have this complement of "colors".

This assignment is a convenient means of segregating the filtering operations, especially in a parallel environment, so that, even when the aspect tensor changes in space, no filtering operation need interfere with another at a point that both share.

(a)


-5,6
-1,6
1,6
5,6
-6,5
-4,5
-3,5
-2,5
-1,5
1,5
2,5
3,5
4,5
6,5
-5,4
-3,4
-1,4
1,4
3,4
5,4
-5,3
-4,3
-2,3
-1,3
1,3
2,3
4,3
5,3
-5,2
-3,2
-1,2
1,2
3,2
5,2
-6,1
-5,1
-4,1
-3,1
-2,1
-1,1
0,1
1,1
2,1
3,1
4,1
5,1
6,1
-1,0
1,0

(b)



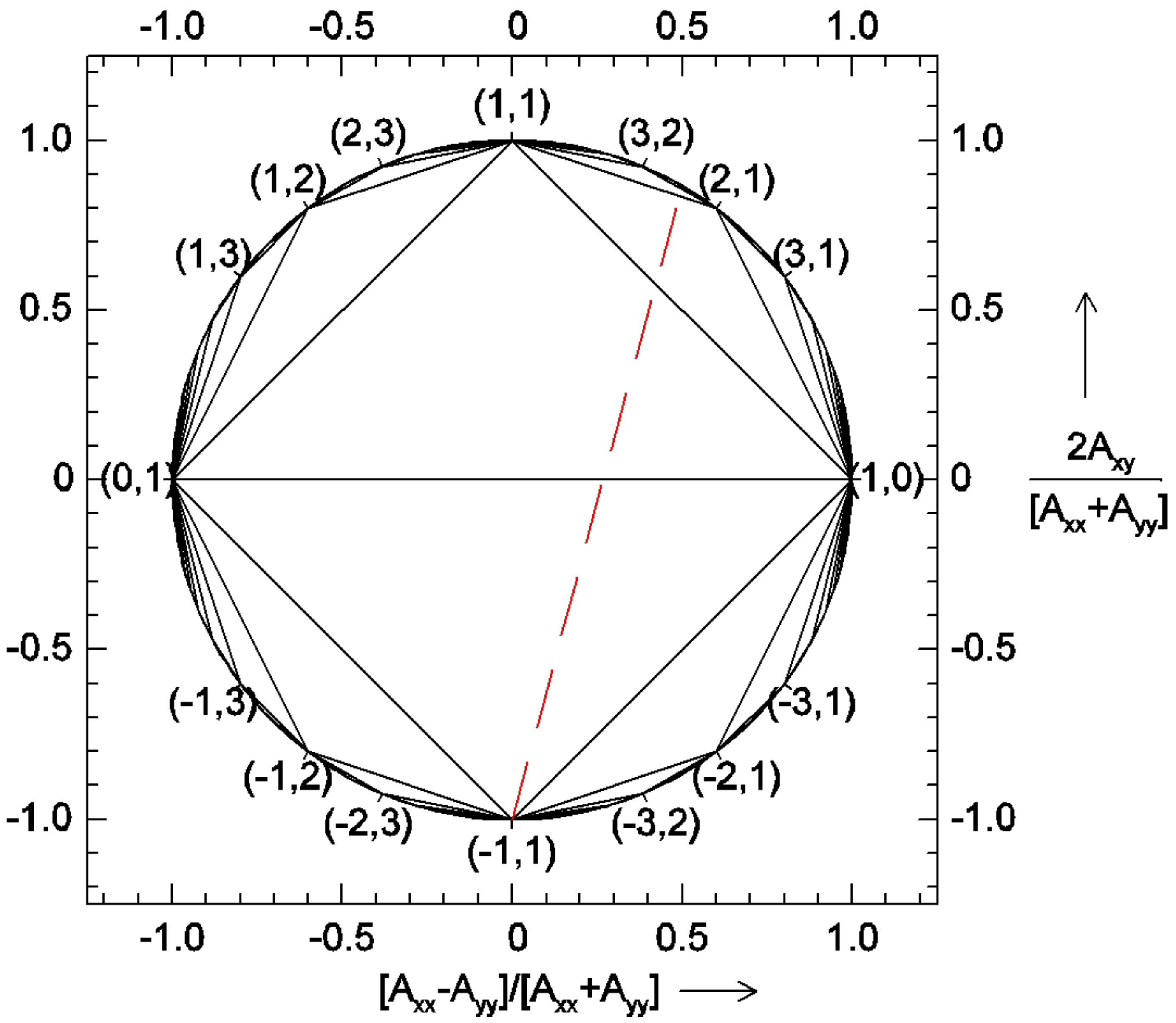
(1,1)
(3,4)
(2,3)
(3,5)
(4,3)
(3,2)
(5,3)
(1,2)
(2,1)
(2,5)
(5,2)
(1,3)
(3,1)
(1,4)
(4,1)
(0,1)
(1,0)
(-1,4)
(-4,1)
(-1,3)
(-3,1)
(-2,5)
(-5,2)
(-1,2)
(-2,1)
(-3,5)
(-2,3)
(-3,4)
(-4,3)
(-3,2)
(-5,3)
(-1,1)

A problem with the basic triad

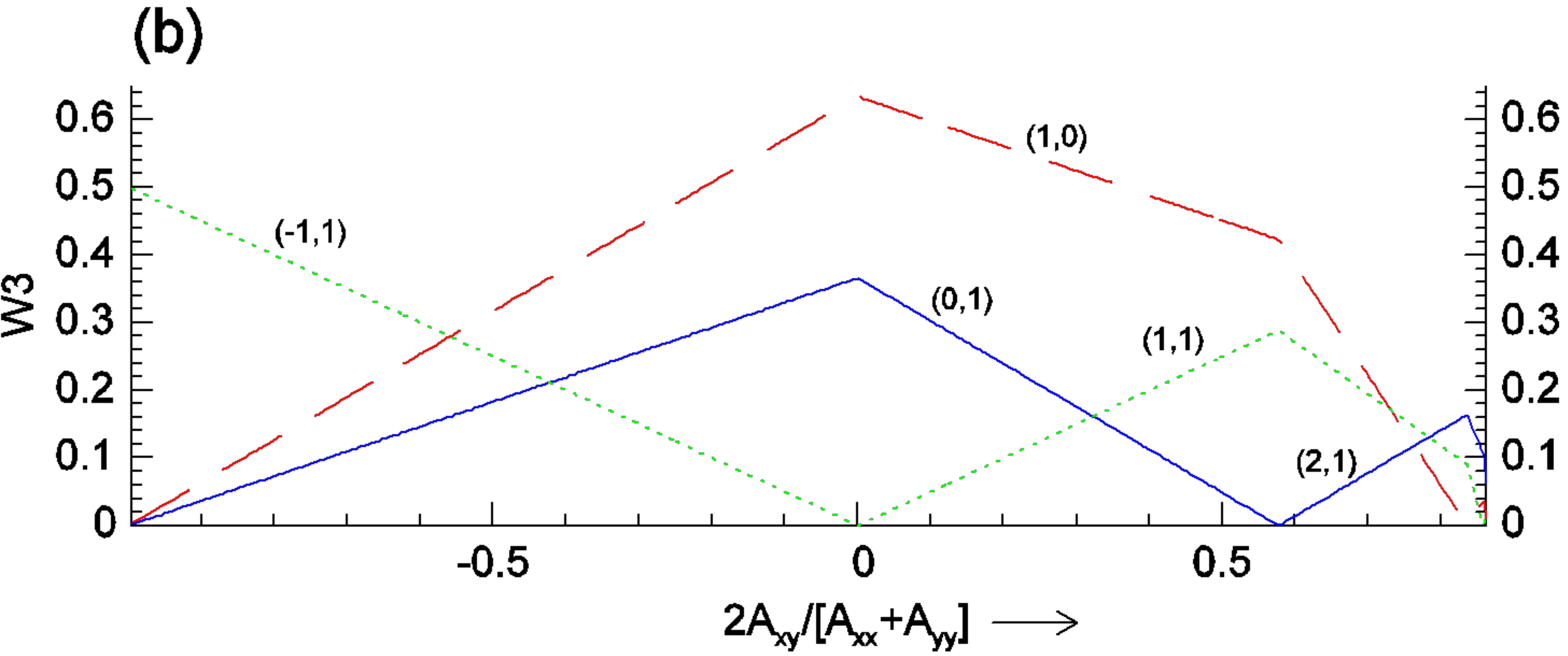
In an inhomogeneous environment, it was noticed that, at geographical locations known to correspond with a transition boundary between two distinct triads, the filters would produce unsightly numerical noise.

Consider the following example, where a transit through the aspect cone is given by a chord in the Klein representation cutting several distinct triad tiles.

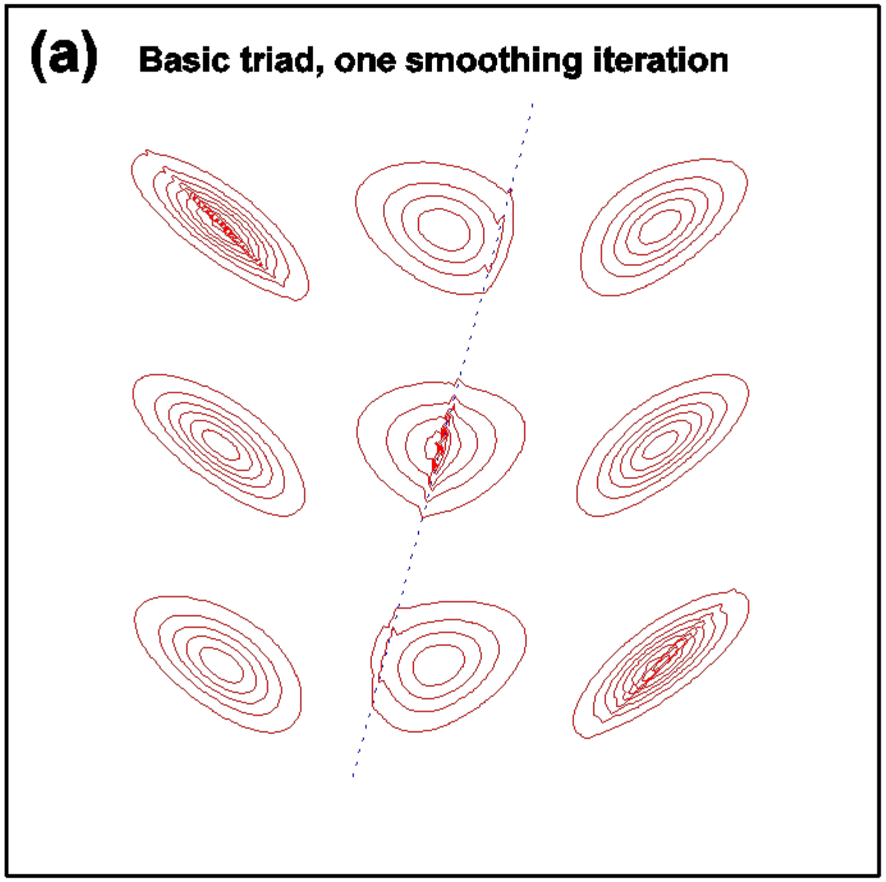
(a)

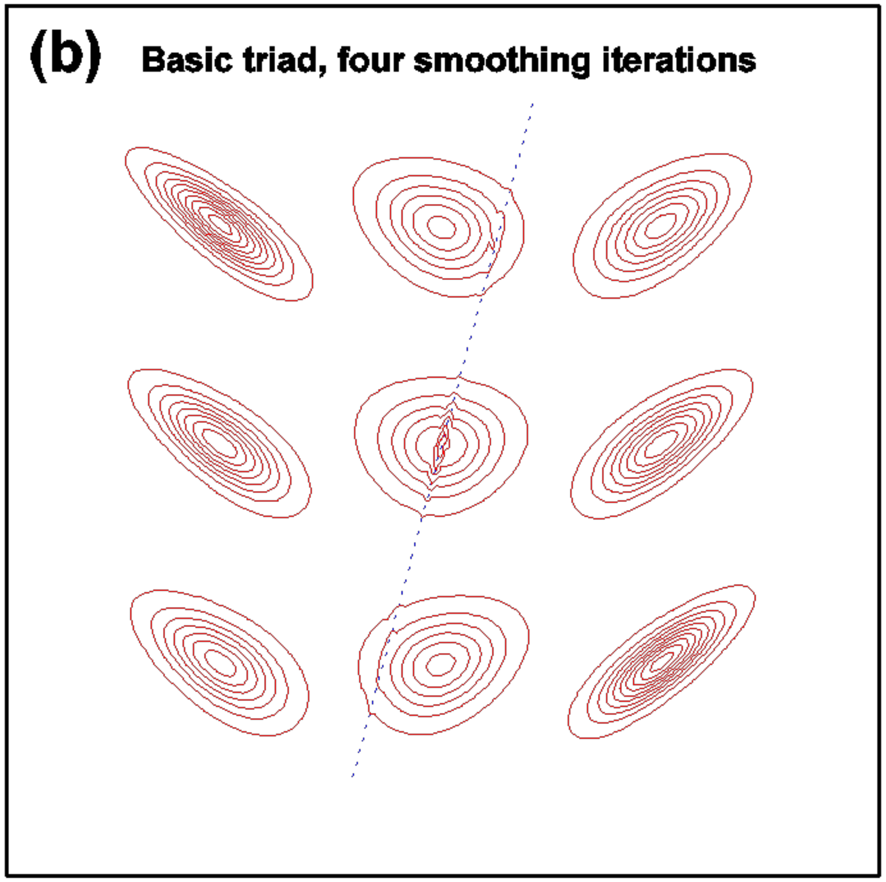

-1.0
-0.5
0
0.5
1.0
(1,1)
(2,3)
(3,2)
(1,2)
(2,1)
(1,3)
(3,1)
(0,1)
(1,0)
(-1,3)
(-3,1)
(-1,2)
(-2,1)
(-2,3)
(-3,2)
(-1,1)
$\frac{2A_{xy}}{[A_{xx}+A_{yy}]}$
$[A_{xx}-A_{yy}]/[A_{xx}+A_{yy}]$

The filtering weights (shown below) are certainly continuous:



But noise is produced nevertheless.





The problem occurs because the “weights” are essentially proportional
To the SQUARE of the characteristic smoothing distance.

These weights were found to rise from and descend to zero at a
linear rate. At small values, this means that the characteristic scale
exceeds the distance to the transition point itself – it is a bit like a
wave “breaking”.

The solution requires us to find a way to cause the weights to
approach zero at a smoother rate.

Again, by exploiting effective linearity of 2nd moments under
sequential composition of the filters, we can “blend” neighboring
Triads to resolve this difficulty. The price we pay is now requiring
(in general) FOUR (instead of just three) line filters at each place.

How do we “blend” triads?

We exploit the additivity of aspect tensors under composition (eg, by sequential application) of their associated Gaussian filters. If we can “compose” a finite number of filters like this, then why not an infinite number? (…at least in principle.)

We could replace a single aspect tensor “point” by any concentric sphere, or spherical shell, of’ points with the same total weight, and the integrated accumulation of aspect tensor components would correspond.

The “weights” of a triad are also linearly distributed through the interior of any given triad. This means that, if we have any weighted distribution of aspect tensors within a given triad, their accumulated aspect tensor is identical to the aspect tensor at the centroid of that distribution, weighted by the total weight of the distribution.

Suppose a “ball” of weighted points, weights symmetrically distributed, replaces the single point aspect tensor. Then wherever that ball intercepts a triad, replace the intersecting part of the ball by the centroid of this portion, appropriately weighted by the weight of this portion. The three line-filters of this triad act like “coordinate bases” in which to “expend” this centroid’s weighted aspect tensor. That is, we are defining the three smoothing “weights” representing this triad. Repeat for all the triads intercepted. Combine the filters implied by all the intersected triads in this way and the resulting composition of filters will have exactly the same aspect tensor as the single-point aspect we started from. We have “blended triads” in a way that now combines (in general) more than one triad (three line filters).

If the “ball” used to perform this blending trick has a radius and radial distribution of weight that varies only smoothly with the position (in aspect space) of the center of the ball, then the weights associated with each line filter will change more smoothly with ball-center-position than do the weights associated directly with the (changing) triad of the ball center by itself.

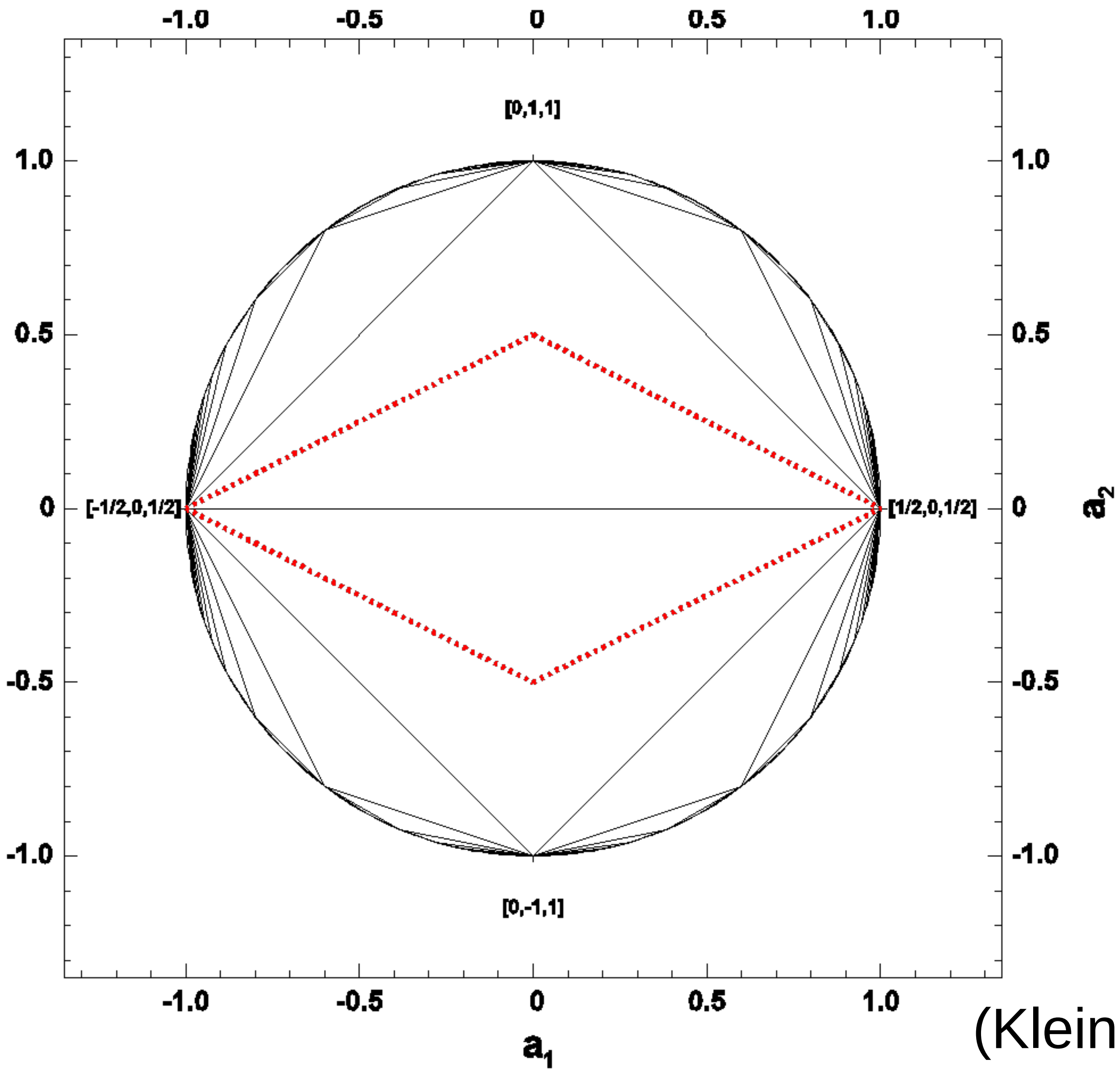
Blending really implies smoothing.

In practice, we wish to blend maximally, within the constraint that the total number of line smoothers implied by the blending does not exceed the minimum bound that is consistent with an effective amount of weight-smoothing.

In the case of the triads of 2D filtering, this bound is four and the number of triads blended is just two.

The “cluster” of two triads is the adjacent pair of triads that accommodates the largest-radius ball centered on the target aspect tensor point.

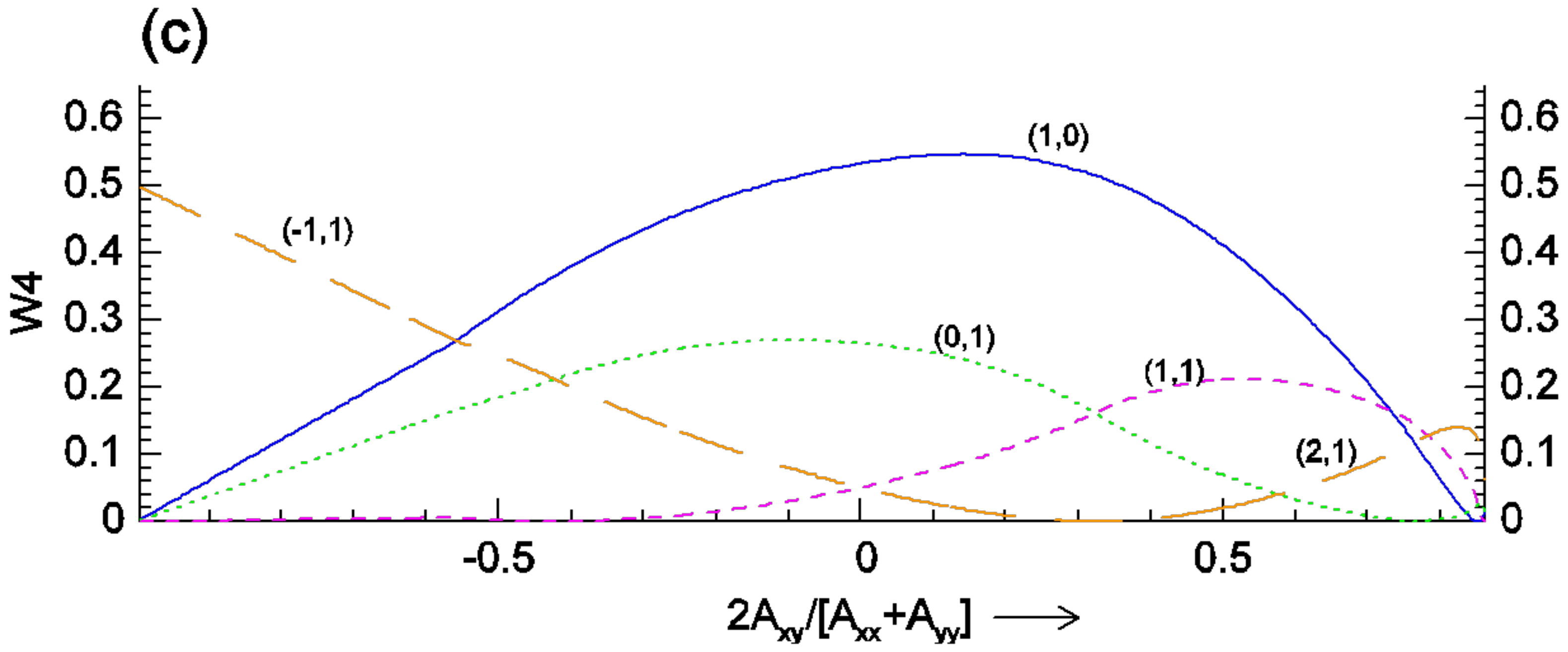
The cluster is centered on a “junction” between the triads involved.

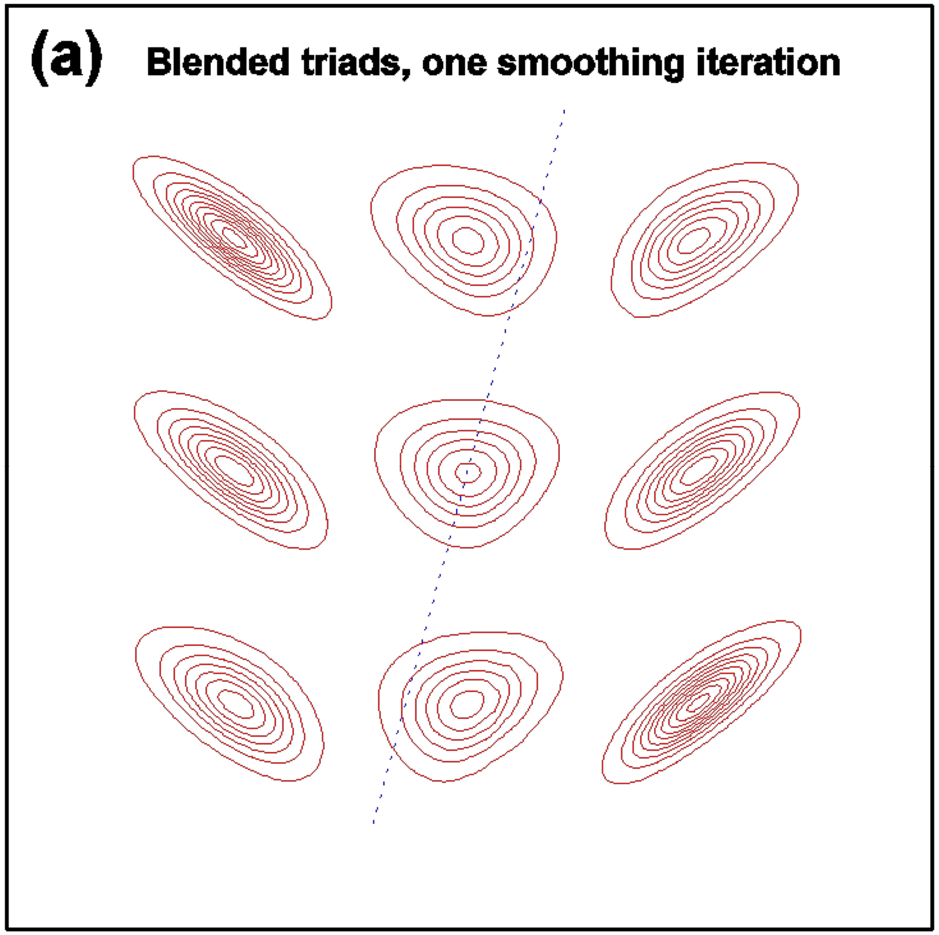


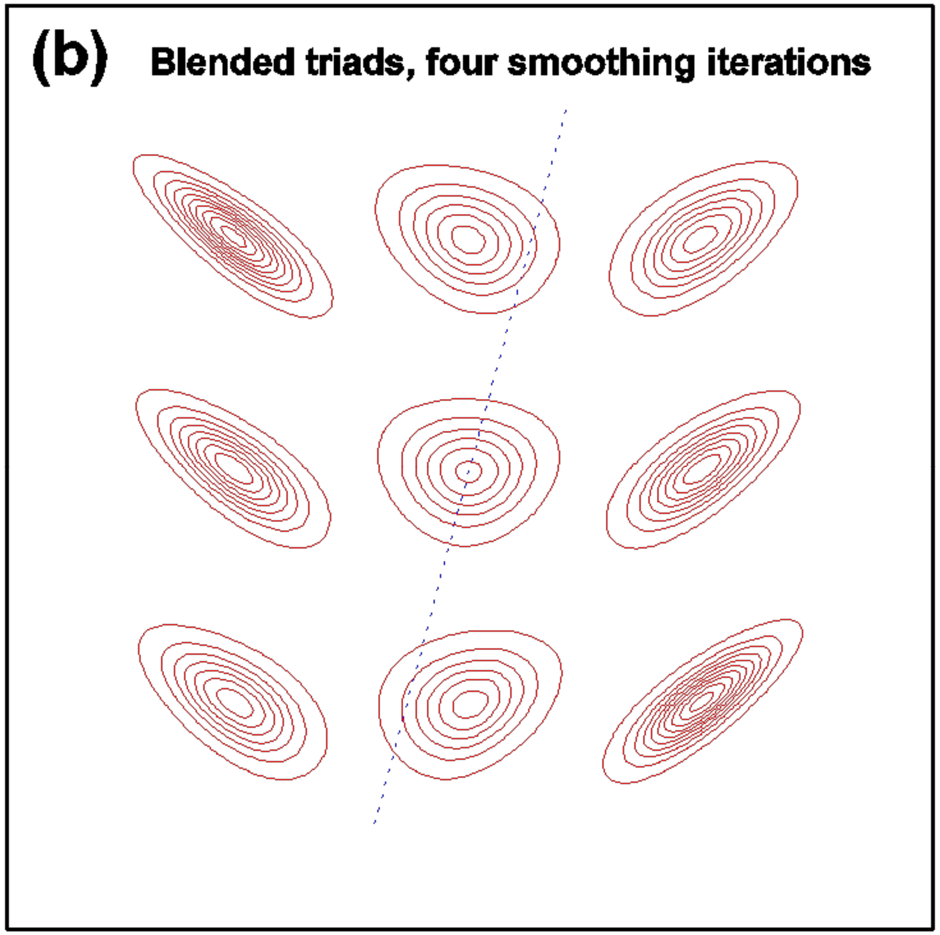
(Klein map)

Apply progressive blending of the **TWO** triads’ weights within red border.

In the case of the straight transit through the aspect cone, the profiles of the blended weights shows that, at least in the proper interior, the weights do go to zero smoothly.



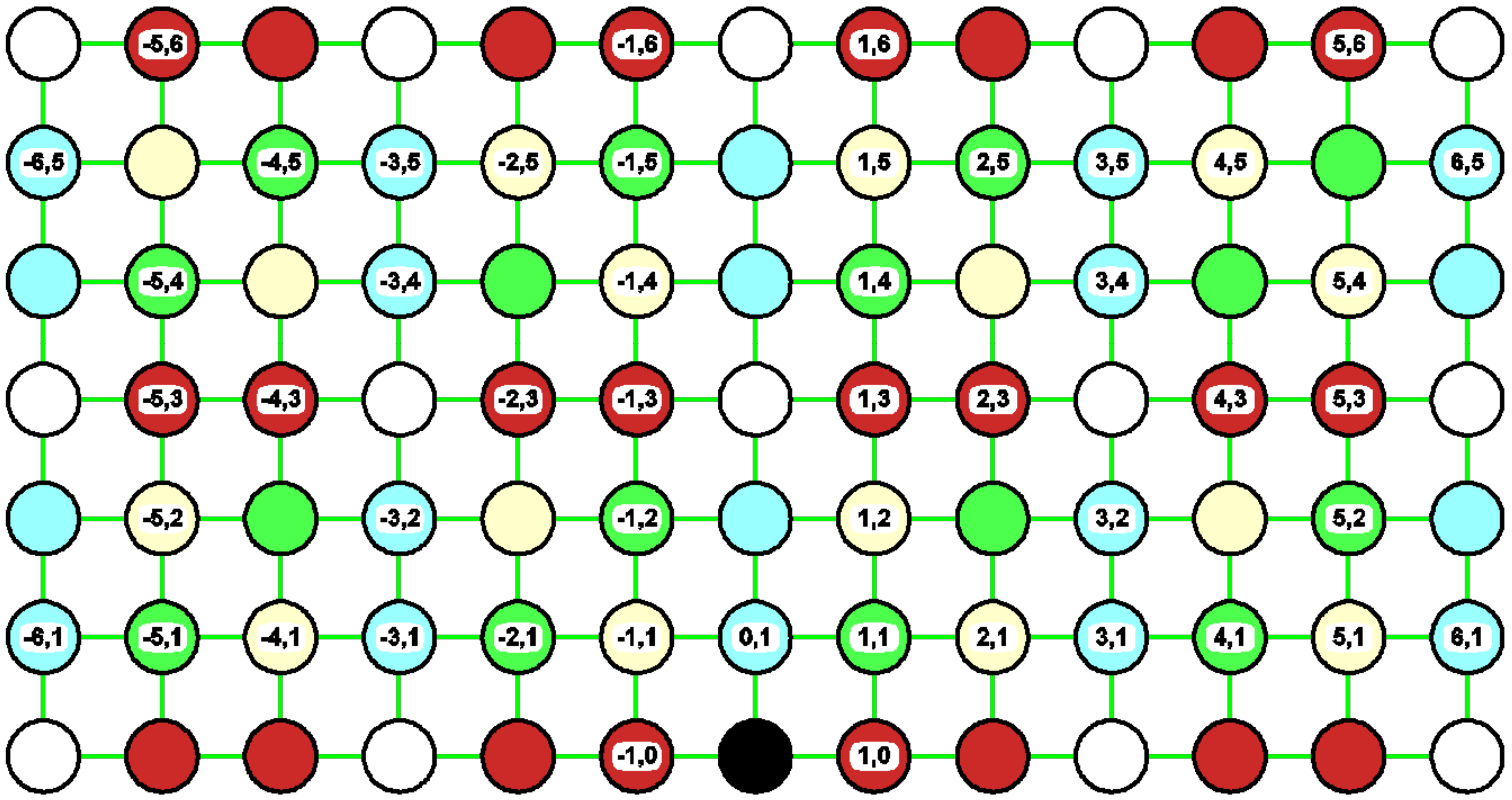


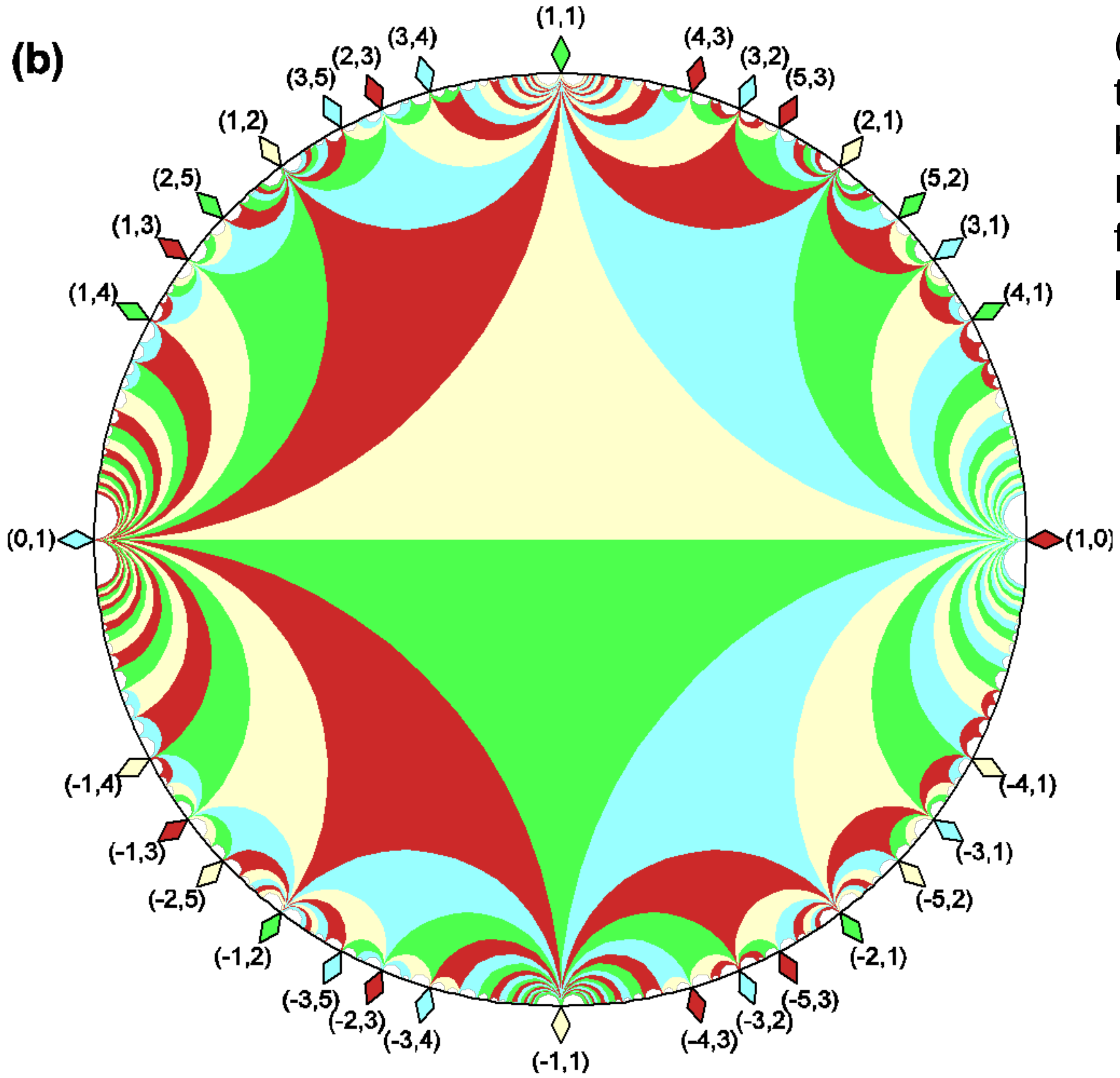


Problem solved!

The color-coding needed to segregate filters for the blended triads:

(a)





(Each triad tile is colored by the color NOT accounted for by the 3 line filters.)

3D: The HEXAD algorithms.

Six independent components are needed to fully specify the aspect tensor in 3D. By analogy with the triad algorithm, we therefore expect six sequential line smoothers, with associated smoothing “weights” to be necessary and sufficient to generate a Gaussian with any prescribed valid aspect tensor.

But what is the appropriate configuration of lines of a basic hexad?

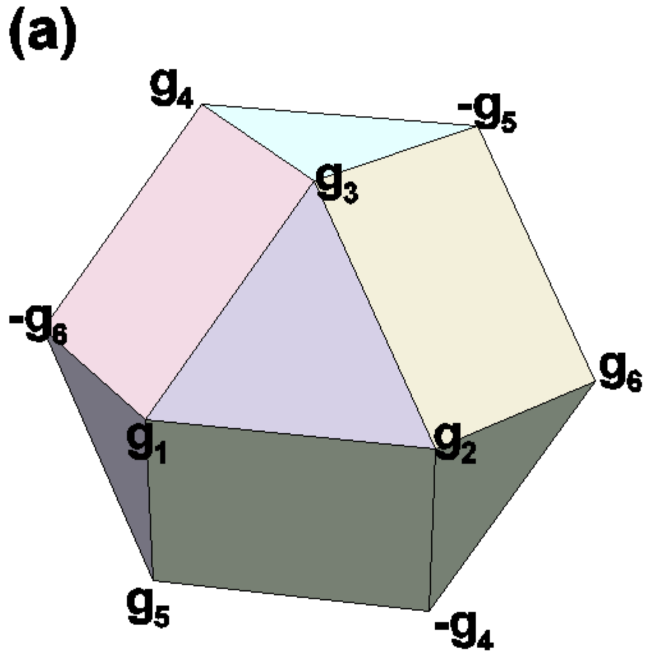
There is a systematic resolution of such a question in n dimensions; just project ALL line generators, assigned equal (unit) “weight”, to aspect space, and take the CONVEX HULL of the set of their images. The boundary of this set is a convex polyhedral shell. A given aspect tensor centrally-projects to a polyhedral “facet” of this shell, and the necessary line filters are taken from those associated with the vertices of this facet.

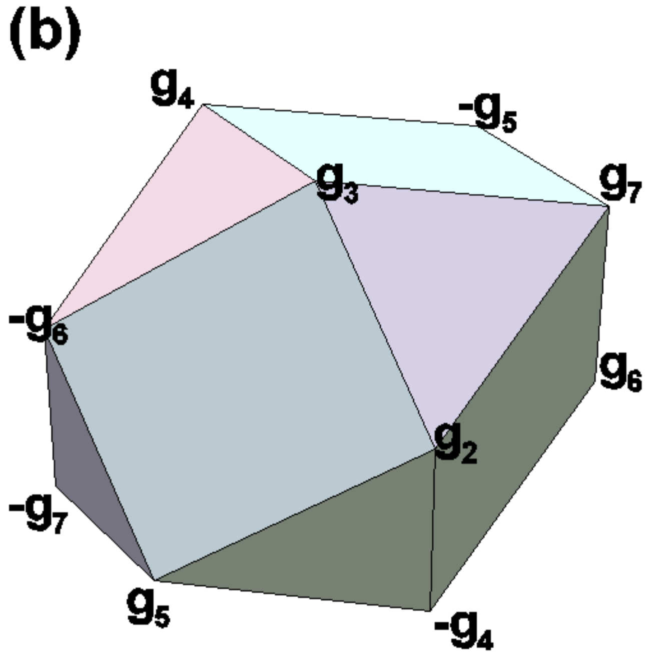
In n > 3 spatial dimensions, there can be more vertices to the lucky facet than there are degrees of freedom [n(n+1)/2] in the aspect tensor. But there is always (at least) one “simplex” subset of n of the facet’s vertices such that the simplex contains the projected aspect tensor. (In 4D, some facets have 12 vertices, while the aspect tensor only needs 10 numbers, for example).

However, in 3D, everything is simple: the convex shell is made up of congruent simplex “tiles” each (being simplexes) having just the desired number, six, vertices.

What is the pre-image of each hexad simplex? I.e., what is the configuration of six line-generators that we associate with a valid “Hexad”?

Geometrically, the generators, together with their negatives, form the 12 vertices of a linear deformation of a “CUBOCTAHEDRON” that contains no other generator.



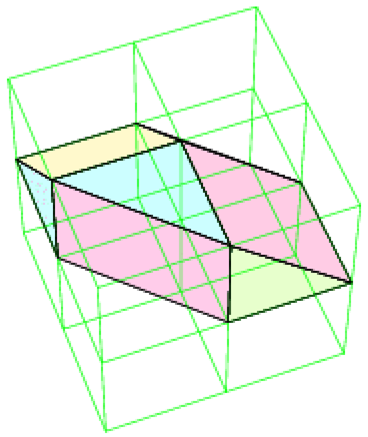


The configuration of line-generators (and their negatives) for the basic "Hexads" used in a 3D lattice to provide anisotropic covariances.

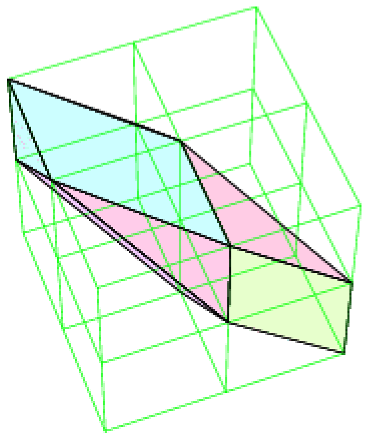
The hexad of generators always forms a (deformed) **CUBOCTAHEDRON** when we fill in the "CONVEX HULL" of generators and their negatives.

The pair shown represent neighboring hexads --- they share five of their generators, but only (a) possesses g1, while only (b) possesses g7.

(a)



(b)

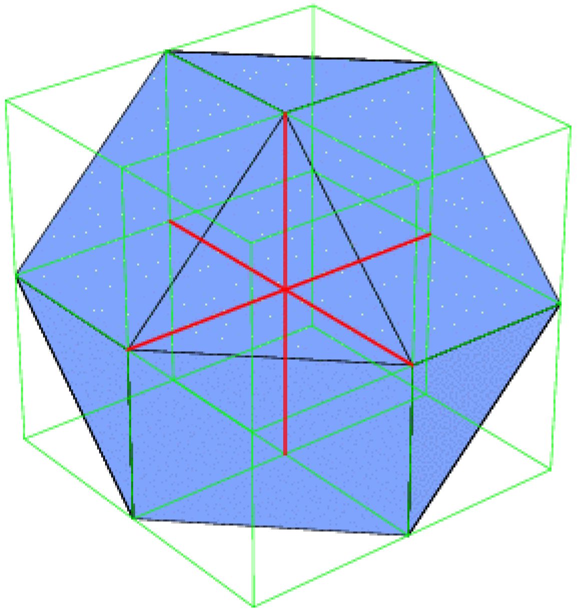


Naturally, the necessity of “blending” is as important for hexads as it is for the triads. But it is a far more complicated problem, since the dimensionality of the projective aspect space is now 6-1 = 5.

The “junction” at the center of the “cluster” is, in this case, the configuration where only three smoothing weights are non-zero, but in such a way that the aspect tensor remains non-degenerate.

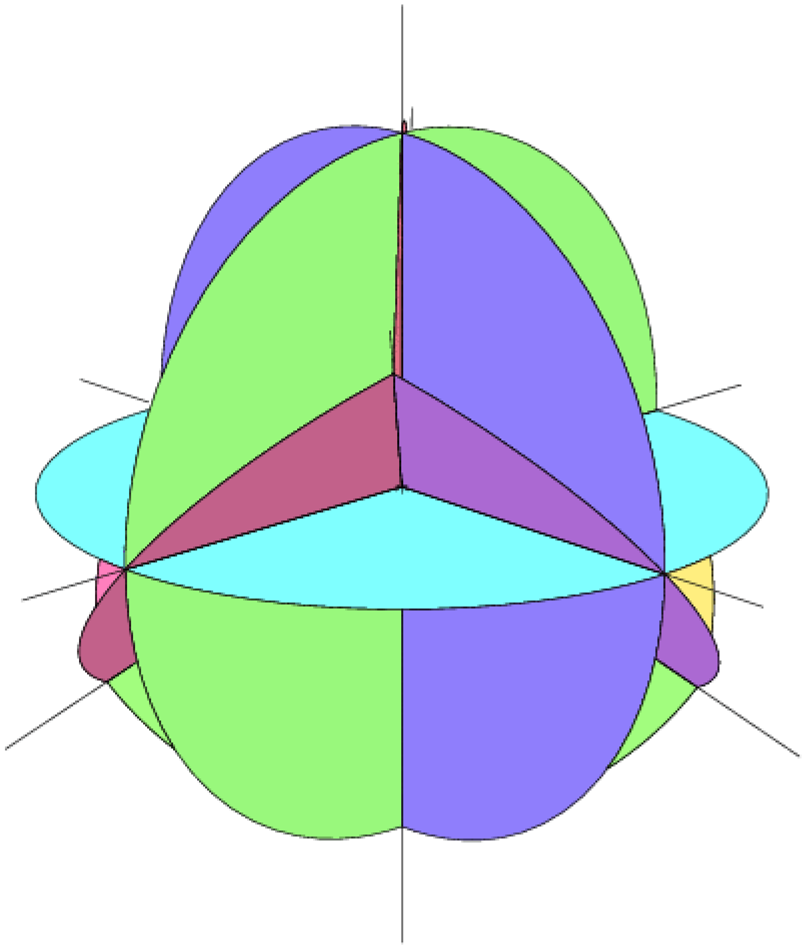
The canonical example is where the three active directions are the three Cartesian axis directions of the cubic lattice.

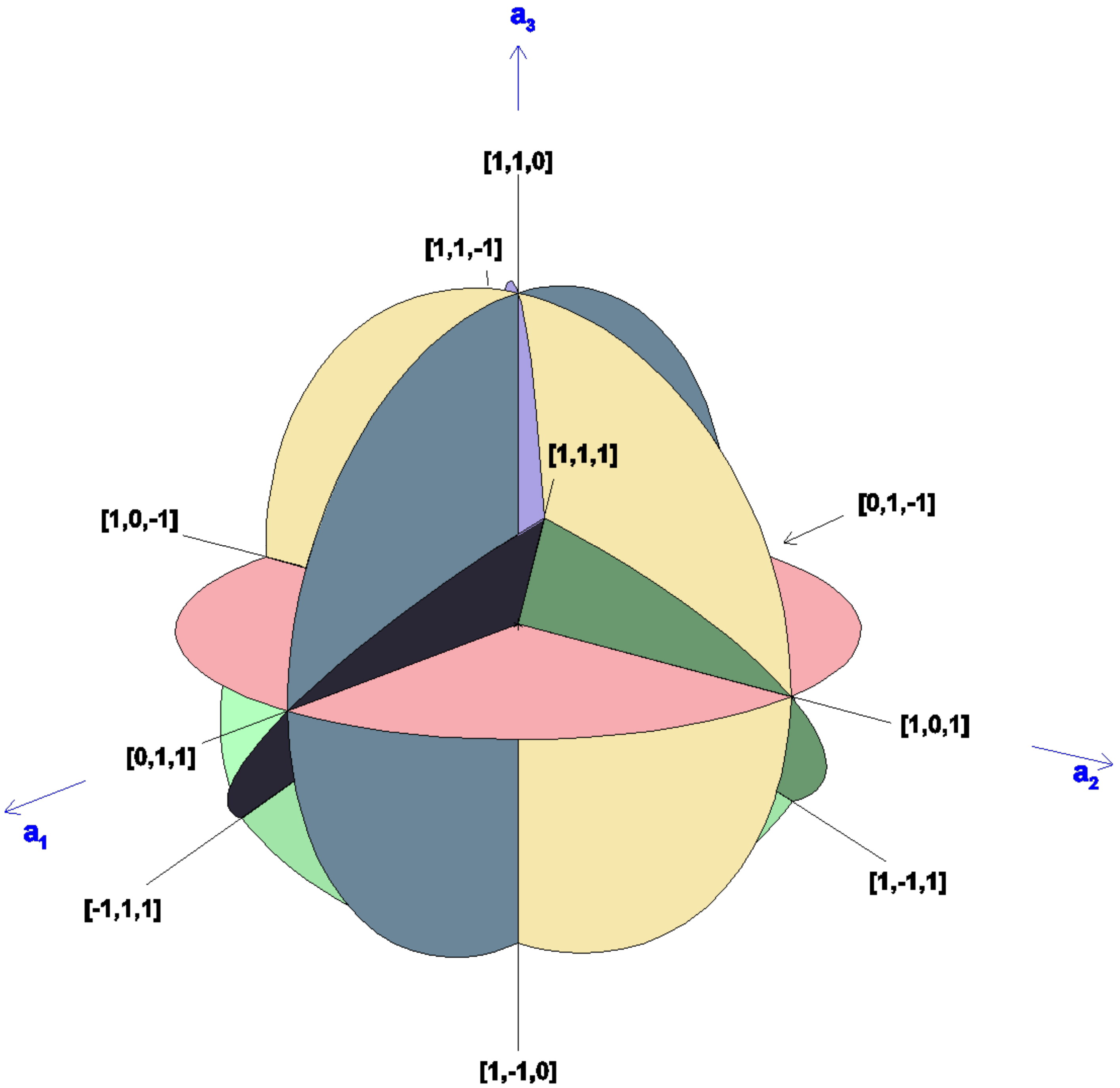
The “junction” is in a three-dimensional linear subspace of the six-dimensional aspect space. The cluster of hexads that surrounds it amounts to 16 hexads. Here they are:



As noted, the junction in this case is three-dimensional, but the interesting structure is in the 3 co-dimensions.

A section through the junction oriented in the most symmetrical way reduces the junction itself to the central point and the hexads meet there in a configuration that looks like this:




a_3
[1,1,0]
[1,1,-1]
[1,1,1]
[0,1,-1]
[1,0,-1]
[1,0,1]
a_2
[0,1,1]
a_1
[-1,1,1]
[1,-1,1]
[1,-1,0]

A set of 16 hexads that share a common junction is called a "CLUSTER".

"Blending" is being done in this case by surrounding the target aspect tensor with a "spherical shell" of sample points, where the radius of the shell is as large as it can be made while still intersecting ONLY hexads that all belong to one cluster. The hexad weights are then integrated over the entire spherical shell. From the preceding picture, it may be verified that, in addition to three line-generators associated with ALL members of a given cluster, ten additional line-generators are also involved with some members of the cluster, making a total of 13 possible line filters involved in the blending at each geographical location.

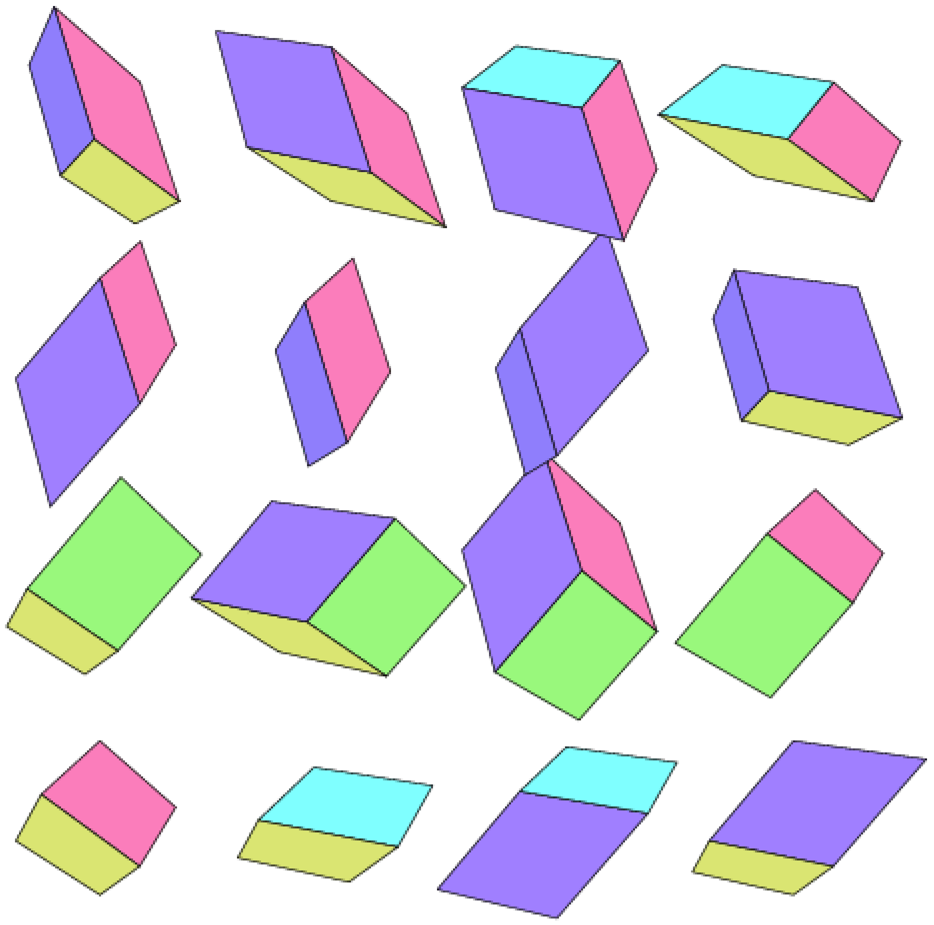
Solving this geometrical problem requires knowing which clusters are contenders. We want the cluster whose boundaries are the most distant, just as in the triads case.

In the space of the lattice generators, a cluster is associated with a 2*2*2 “parallelepiped” (we saw this) in the earlier animation of 16 hexads that meet at a junction) ; the 13 generators (and their negatives) form the 26 points that surround the origin.

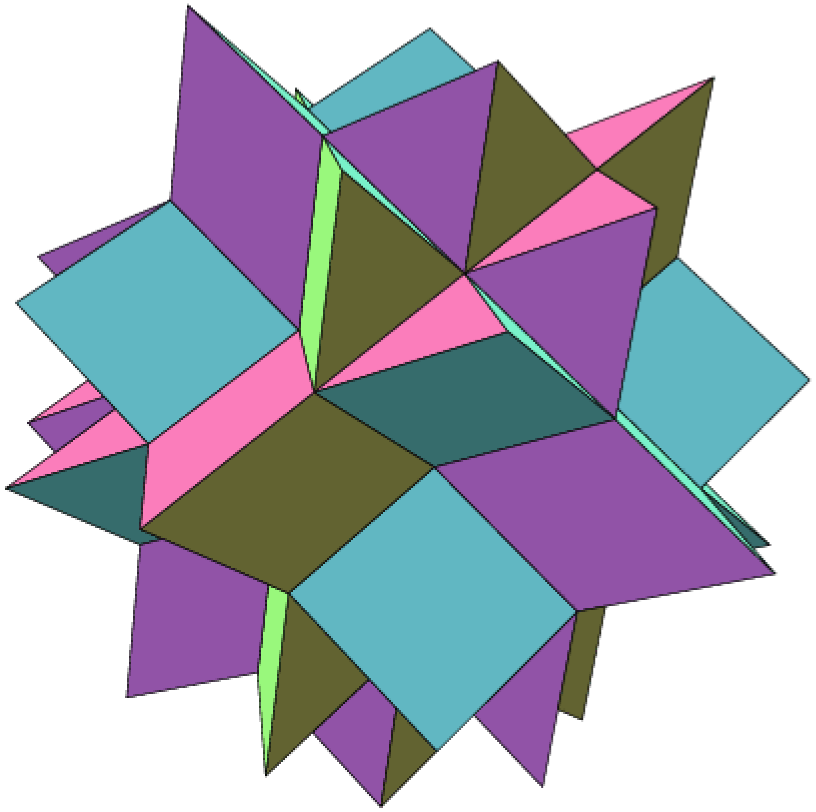
There are 16 clusters that intersect a given aspect tensor and that have the basic hexad in common. These are the contenders that need to be checked.

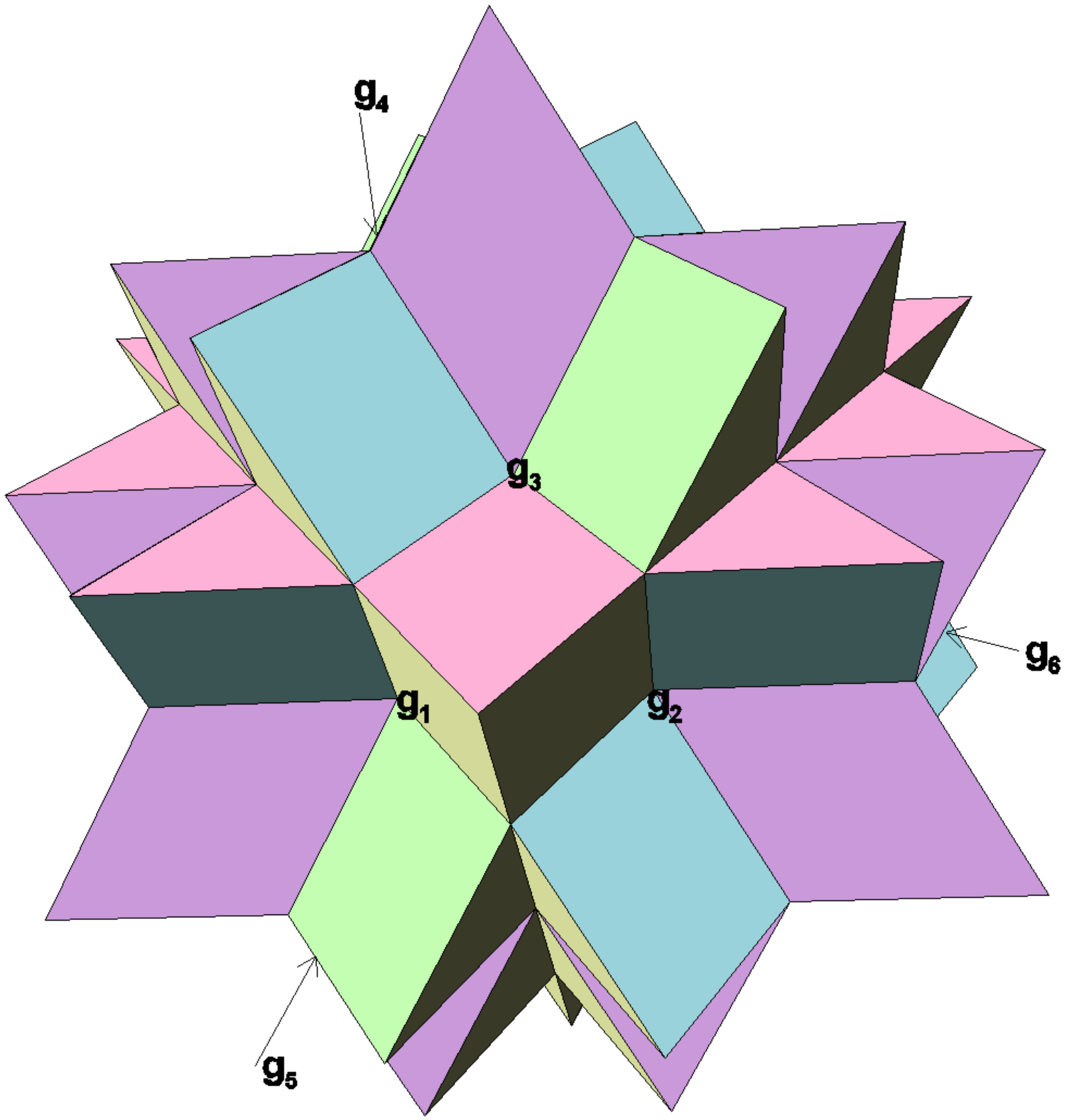
The most symmetrical pictures of the set of 16 clusters are obtained, in physical lattice space, when the lattice is of the closest-packing “face-centered cubic” form.

Here are the pre-images (lattice space) of the “clusters”, depicted as their 2*2*2 parallelepipeds:



In terms of the generator-lattice parallelepipeds, the picture we obtain from the union of these 16 parallelepipeds is the “starburst”:




g_4
g_3
g_6
g_1
g_2
g_5

Armed with a geometrical appreciation of how the hexads fit together in six-dimensional aspect space, it is now possible to carry out a systematic computation of the blending weights.

In this case, the “ball” of points surrounding the target aspect point is given a radial profile that makes the computation of these weights relatively easy.

The best radial profiles to use belong to the family of “Beta distributions”. See NCEP Office Note 447 for details.

The tabulation of the blending weights is actually being done “off-line”.
the necessary tables are five-dimensional, but highly symmetrical
(which saves a lot of storage).

The overall shape of the table is a “simplex”, but symmetry reduces
the non-redundant portion to a fragment of some other polyhedral shape
(unknown) determined by a bunch of inequalities.

At the present time, code to generate the table entries at a sufficient
resolution is being tested. It will involve several thousand table locations,
and each location will need to store 13 normalized blending weights, together
with an index notation to prescribe exactly which line directions are
Involved.

As in the triads examples, the basic hexad and the blended hexad algorithms
are both associated with a color scheme. The basic hexad method needs
seven colors. The blended scheme requires 13, exactly in keeping with
the 13 line directions that belong to a given cluster.

Other refinements

- Multigrid option to allow non-Gaussians
- Reformulation of filters in Riemann space
- New normalization in Riemann space
- Nested grids treatment of global scales
- New treatment of parallelization?

Possible future considerations

Extend the basic and blended algorithms to four dimensions

Adopt vorticity and divergence as control variables (instead of stream function and velocity potential. (Multigrid structure might help with the needed inversions). Vorticity and divergence patterns are more likely to be quasi-conservative as 4D tracers.

Attempt to represent the analysis~background error ratio in terms of synthetic recursive filters. This would help to characterize the analysis error better, and would also provide a tool for better preconditioning.

Thank you for your attention.